

I n s i d e Q u i c k T i m e

API Reference

For QuickTime 5



April 2001

 Apple Computer, Inc.

© 2001 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc., except to make a backup copy of any documentation provided on CD-ROM.

The Apple logo is a trademark of Apple Computer, Inc.
Use of the “keyboard” Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for typographical errors.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Macintosh, Mac, and QuickTime are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD “AS IS,” AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Preface	About This Reference	15
Volume I: Functions A–H		19
Functions: A		21
Functions: B		52
Functions: C		58
Functions: D		166
Functions: E		314
Functions: F		344
Functions: G		380
Functions: H		666
Volume II: Functions I–Q		669
Functions: I		671
Functions: L		779
Functions: M		784
Functions: N		1019
Functions: O		1126
Functions: P		1136
Functions: Q		1155
Volume III: Functions R–Z and Callbacks		1449
Functions: R		1451
Functions: S		1526
Functions: T		1916
Functions: U		1979
Functions: V		1985
Functions: W		2061
Functions: X		2062

Callbacks	2075
Volume IV: Types and Constants	2157
Data Structures	2159
Atoms	2507
Defined Data Types	2619
Atom Types	2619
Codec Types	2619
Component Types	2621
Data Handling Types	2622
Effects Types	2622
Endian Types	2623
File System Types	2624
Graphics Types	2624
Media Handling Types	2626
Memory Types	2627
Movie Controller Types	2628
Movie Types	2628
Music Types	2630
Numeric Types	2631
Sequence Grabbing Types	2632
SMIL Types	2633
Sound Types	2633
Sprite Management Types	2636
Streaming Types	2636
String Types	2637
System and Toolbox Types	2638
Text Types	2640
Timing and Callback Types	2640
Universal Procedure Pointers	2641
Video Types	2646
Virtual Reality Types	2647
Windows Programming Types	2648

Constants	2649
Atom ID Codes	2649
Codec Identifiers	2655
Color Constants	2657
Component Identifiers	2657
Controller Numbers	2658
Data References	2661
Effects Codes	2662
Error Codes	2663
File Types and Creators	2668
Graphics Transfer Modes	2670
Localization Codes	2673
Movie Controller Actions	2680
Music Controllers	2682
Parameter Behaviors	2685
Pixel Formats	2686
QTMA Events	2686
QTVR Cursors	2688
Sound Commands	2691
Sound Device Features	2692
Sound Formats	2692
Sound Information Selectors	2693
Sound Sample Rates	2695
Streaming Transport Atoms	2695
User Data Identifiers	2696
Video Digitizer Capabilities	2696

Volume V: Programming Summary and Indexes 2705

About Volume V	2707
Using Movie Toolbox Data Structures	2709
Movie Identifiers	2709
The Time Structure	2710
The Fixed-Point and Fixed-Rectangle Structures	2710
The Sound Description Structure	2711

Getting and Playing Movies	2711
Initializing the Movie Toolbox	2712
Error Functions	2712
Movie Functions	2713
Saving Movies	2715
Controlling Movie Playback	2717
Using the Full Screen	2717
Controlling Hardware Scaling	2718
Movie Posters and Movie Previews	2718
Movies and Your Event Loop	2720
Preferred Movie Settings	2721
Enhancing Movie Playback Performance	2722
Disabling Movies and Tracks	2724
Generating Pictures From Movies	2725
Creating Tracks and Media Structures	2726
Working With Progress and Cover Functions	2726
Modifying Movie Properties	2728
Working With Movie Spatial Characteristics	2728
Working With Sound Volume	2732
Working With Movie Time	2732
Working With Track Time	2733
Working With Media Time	2735
Finding Interesting Times	2735
Locating a Movie's Tracks and Media Structures	2736
Working With Track References	2737
Working With Alternate Tracks	2738
Working With Track Sound	2740
Working With Data References	2740
Determining Movie Creation and Modification Time	2741
Working With Media Samples	2742
Working With Movie User Data	2743
Managing the Video Frame Playback Rate	2745

Editing Movies	2746
High-Level Movie Editing Functions	2746
Undo for Movies	2748
Low-Level Movie Editing Functions	2749
Editing Tracks	2750
Undo for Tracks	2751
Adding Samples to Media Structures	2752
Manipulating Media Input Maps	2753
Interacting With Media Handlers	2753
Selecting Media Handlers	2754
Working With Media Handler Properties	2755
Video Media Handler Functions	2755
Sound Media Handler Functions	2755
Text Media Handler Functions	2756
Previewing Files	2757
Creating File Previews	2757
Displaying File Previews	2758
Working With Time Bases	2759
Creating and Disposing of Time Bases	2759
Working With Time Base Values	2760
Working With Times	2762
Time Base Callback Functions	2763
Working With Matrices	2764
Managing Matrices	2764
Applying Matrix Transformations	2766
Working With QT Atoms	2766
Creating and Disposing of Atom Containers	2766
Creating New Atoms	2767
Copying Existing Atoms	2767
Retrieving Atoms and Atom Data	2768
Modifying Atoms	2769
Removing Atoms From an Atom Container	2770

Working With Access Keys	2770
Registering and Unregistering Access Keys	2771
Retrieving Access Keys	2771
Managing Progressive Downloads	2772
High-Level Download Control	2772
Low-Level Download Control	2772
Using Movie Controllers	2773
Movie Controller Actions	2773
Associating Movies With Controllers	2774
Managing Controller Attributes	2775
Handling Movie Events	2776
Editing Movies With a Controller	2777
Getting and Setting Movie Controller Time	2778
Customizing Event Processing	2779
Managing Components	2780
Component Data Structures	2780
Component Functions Used By Applications	2781
Functions Used By Components	2782
Managing Image Compression	2786
Image Compression Data Structures	2786
Getting Information About Compressed Data	2787
Getting Information About Compressor Components	2788
Working With Pixel Maps	2789
Working With Image Descriptions	2790
Working With Pictures and PICT Files	2790
Making Thumbnail Pictures	2791
Working With Sequences	2792
Changing Sequence-Compression Parameters	2794
Constraining Compressed Data	2795
Changing Sequence-Decompression Parameters	2795
Working With the StdPix Function	2797
Aligning Windows	2797
Working With Graphics Devices and Graphics Worlds	2798

Image Compression Progress and Completion Callbacks	2799
Image Compression Manager Utility Functions	2799
Using Image-Compression Dialogs	2800
Getting Default Settings for an Image or a Sequence	2800
Displaying the Standard Image-Compression Dialog Box	2801
Compressing Still Images	2801
Compressing Image Sequences	2802
Working With Image or Sequence Settings	2802
Specifying a Test Image	2803
Positioning Dialog Boxes and Rectangles	2803
Creating a Compression Graphics World	2804
Using the Base Image Decompressor	2804
Base Image Decompressor Data Structures	2804
Base Image Decompressor Functions	2805
Using Data Codec Components	2807
Data Codec Functions	2807
Sequence Grabbing	2808
Sequence Grabber Data Structures	2808
Configuring Sequence Grabber Components	2809
Controlling Sequence Grabber Components	2811
Working With Sequence Grabber Settings	2812
Working With Sequence Grabber Characteristics	2813
Working With Sequence Grabber Outputs	2814
Working With Channel Characteristics	2815
Working With Channel Devices	2818
Working With Video Channels	2819
Working With Video Fields	2821
Working With Sound Channels	2821
Video Channel Callback Functions	2823
Using Sequence Grabber Channels	2824
Configuring Sequence Grabber Channel Components	2824
Controlling Sequence Grabber Channel Components	2824
Configuration Functions for All Channel Components	2826

Managing Channel Devices	2828
Configuration Functions for Video Channel Components	2828
Configuration Functions for Sound Channel Components	2831
Utility Functions for Sequence Grabber Channel Components	2833
Using Sequence Grabber Panels	2835
Managing Your Panel Component	2835
Processing Your Panel's Events	2836
Managing Your Panel's Settings	2837
Working With Data Handlers	2838
Selecting a Data Handler	2838
Identifying Data References	2839
Reading Movie Data	2839
Writing Movie Data	2840
Managing Data Handler Components	2841
Using Video Digitizers	2842
Video Digitizer Data Structures	2842
Getting Information About Video Digitizer Components	2843
Setting Source Characteristics	2843
Selecting an Input Source	2844
Setting Video Destinations	2845
Controlling Compressed Source Devices	2846
Controlling Digitization	2848
Controlling Color	2849
Controlling Analog Video	2850
Selectively Displaying Video	2852
Video Clipping	2853
Video Digitizer Utilities	2853
Exchanging Movie Data	2855
Importing Movie Data	2855
Configuring Movie Data Import Components	2856
Exporting Movie Data	2858
Configuring Movie Data Export Components	2859
Exporting Text	2860

Using Derived Media Handlers	2861
Managing Your Media Handler Component	2861
General Data Management	2861
Managing Graphics Data	2864
Managing Sound Localization	2866
Making a Progressive Download Time Table	2866
Reporting Capabilities to the Base Media Handler	2866
Using Clock Components	2867
Getting the Current Time	2867
Using Callback Functions	2867
Managing the Time	2868
Movie Toolbox Clock Support Functions	2869
Using the Timecode Media Handler	2869
Working With the Timecode Media Handler	2870
Using Preview Components	2872
Displaying Previews	2872
Handling Preview Events	2872
Creating Previews	2873
Transcoding Images	2873
Image Transcoder Support	2873
Using Text Channel Components	2874
Text Channel Support	2874
Tweening	2875
Tween Component Requirements	2876
Using Graphics Importers	2876
Managing Graphics Importers	2876
Obtaining a Graphics Importer Instance	2878
Getting Image Characteristics	2878
Setting Drawing Parameters	2879
Drawing Imported Images	2880
Saving Image Files	2881
Getting MIME Types	2881

Specifying a Graphics Import Data Source	2882
Retrieving Graphics Import Image Data	2883
Using Graphics Exporters	2883
Setting the Input and Output of a Graphics Exporter Instance	2883
Internal Graphics Export Routines	2883
Finding Out About Graphics Export Image Formats	2884
Obtaining Graphics Exporter Settings	2885
Accessing Graphics Exporter Settings	2885
Getting and Setting Progress Procs	2887
Specifying Sources for Graphics Exporter Input Images	2887
Restricting the Range of an Input Image's Source	2888
Reading Graphics Exporter Input Data	2889
Accessing a Graphics Exporter's Input Image	2889
Specifying Destinations for Output Images	2890
Writing Graphics Exporter Output Data	2891
Using Video Outputs	2892
Controlling the Video Output Display Mode	2892
Registering the Name of Video Output Software	2892
Controlling Video Output	2893
Finding Components Associated With a Video Output	2894
Saving and Restoring Component Configurations	2894
Using the Standard Sound Dialog	2895
Managing the Sound Dialog	2895
Sound Dialog Selectors	2895
Working With Video Effects	2897
High-Level Effects Functions	2897
Low-Level Effects Functions	2898
Creating an Effect Sample Description	2898
Working With the Vector Codec	2899
Vector Codec Component Functions	2899
Working With the Sprite Toolbox	2900
Working with Sprite Worlds	2900
Creating and Manipulating Sprites	2901

Using the Sprite Media Handler	2902
Managing Movie Sprites	2902
Working With Wired Sprites	2903
Working With Virtual Reality	2904
Initializing and Terminating QuickTime VR	2904
Managing QuickTime VR Movie Instances	2904
Manipulating Viewing Angles and Zooming	2905
Getting Scene and Node Information	2906
Managing Hot Spots	2906
Handling Events	2907
Intercepting QuickTime VR Manager Routines	2908
Managing Object Nodes	2909
Managing Imaging Characteristics	2910
Converting Angles and Points	2911
Managing QuickTime VR Movie Interactions	2912
Determining Viewing Limits and Constraints	2912
Managing VR Memory	2913
Accessing Image Buffers	2914
Programming Music	2914
Using the Tune Player	2914
Allocating and Using Note Channels	2916
Note Allocator Interface Tools	2918
Note Allocator Configuration and Utilities	2919
Managing Synthesizers	2920
Managing Instruments and Parts	2921
Miscellaneous Music Component Functions	2922
Instrument Component Functions	2923
MIDI Component Functions	2924
Importing MIDI Files	2924
Managing the Generic Music Component	2925
Calling Generic Music Component Clients	2925
Index of Data Types	2927
Index of Constants	2953

Functions By Header Files	3005
Bibliography	3053
Glossary	3057
Document History	3061

About This Reference

This multivolume work defines the complete C application programming interface (API) for the QuickTime system software. It's Apple's official guide to the more than 2000 function calls that QuickTime applications can make. The printed edition consists of four volumes internally cross-referenced by page numbers. The online edition consists of a single Adobe PDF file with live internal links and the same content in a set of linked HTML pages posted to the QuickTime Developer Documentation website.

Note

The pages in the printed and PDF versions of this reference are numbered the same and run sequentially from the start of Volume I to the end of Volume V. Every page citation is preceded by its volume number so you know which book to open in the printed version. ♦

QuickTime is a system-level software package that computer users store on their Windows and Macintosh systems. It lets the host computer play and stream movies, synthesize music, display virtual reality environments, and generate animated graphics. Your code can invoke QuickTime's capabilities through the API described in this document.

This reference assumes that you are familiar with programming techniques and with the C programming language. For a more general overview of the QuickTime API see *Discovering QuickTime*, available in computer bookstores.

For further technical information about using the QuickTime API, see the books and websites cited in the bibliography (V-3053). It lists a number of introductory and advanced programming guides designed to help you exploit all the capabilities of this powerful software.

Content Summary

Volumes I through III of this reference list the QuickTime API functions alphabetically, describing how each one works. Where constants are passed

directly to a function, they are explained in the function's description. Many function descriptions are illustrated with code snippets. Callbacks are listed alphabetically at the end of Volume III.

Volume IV describes the data types that the QuickTime API uses—structs, atoms, and defined types. It also documents certain global constants not covered in Volumes I–III.

Volume V contains a programming summary designed to help you understand how the QuickTime API is used to perform actual tasks. Also included are alphabetical cross-indexes of data types and constants, plus an index of functions arranged by their order in the QuickTime header files. A preface at the beginning of Volume V tells you more about these access aids.

At the end of Volume V are a bibliography, a glossary, and a history of the revisions to this reference.

A table of contents that covers all five volumes is included at the beginning of Volume I.

Documentation Principles

The technical documentation sections of this reference follow certain guiding principles:

- **Multiple publication:** Every release of this reference is published simultaneously in paper, PDF, and HTML. The paper version includes a CD that contains the PDF and HTML versions. The content, references, and links are the same in all these formats, so that information found in one medium can easily be located in another.
- **Quick access:** This reference is designed for quick access to information. Functions, callbacks, structs, and atoms are listed alphabetically by name in Volumes I through IV. Constants and data types are grouped logically, but are also indexed alphabetically at the end of Volume IV. All cross-references cite volume and page numbers for users of the paper version but also appear as software links in the PDF and HTML versions.
- **Explanatory closure:** All API elements mentioned in this reference, including all parameter and field types, are linked to their documentation at other locations. The chain of explanations is carried down to the level of C reserved types (Boolean, short, long, etc.).

- **Document history:** Volume IV contains a full history of revisions to this reference, both by date and by API element.
- **Links to programming guides:** This reference is designed to be used in conjunction with the Apple “Inside QuickTime” programming guides described in the bibliography (V–3053). Future Inside QuickTime books will cite this reference for programming details.

Typical Function Description

A typical function description in Volumes I–III looks like this:

GetTimeBaseStatus		
Determines when the current time of a time base would fall outside of the range of values specified by the time base's start and stop times.		Abstract
<pre>long GetTimeBaseStatus (TimeBase tb, // IV-2425 TimeRecord *unpinnedTime); // IV-2209</pre>		Declaration
tb		
The time base for this operation. Your application obtains this time base identifier from the <code>NewTimeBase</code> (II-1172) function.		Term defined in glossary
unpinnedTime		
A pointer to a time structure that is to receive the current time of the time base. Note that this time value may be outside the range of values specified by the start and stop times of the time base.		Parameter descriptions
<i>function result</i> See below.		
Function Result Constants		
timeBaseBeforeStartTime		
Returned if the time value in the time structure referred to by the <code>unpinnedTime</code> parameter lies before the start time of the time base.		Constant descriptions
timeBaseAfterStopTime		
Returned if the time value in the time structure referred to by the <code>unpinnedTime</code> parameter lies after the stop time of the time base.		
Discussion		
The status information returned by this function allows you to determine when the current time of a time base would fall outside of the range of values specified by the start and stop times of the time base. This can happen when a time base relies on a master time base or when its time has reached the stop time.		Discussion
C Interface File		
Movies.h		
Related Java Methods		
quicktime.std.clocks.TimeBase.getStatus()		Other information
See Also		
See <code>GetTimeBaseFlags</code> (I-465).		

Function descriptions provide you with the following programming information:

- The **abstract** describes briefly what the function does or how you use it. Some Apple programming guides repeat these abstracts to jog your memory about a function's role in the API. You could think of a function's abstract as its natural-language declaration.
- The **declaration** schematizes the way you call the function in your code. The parameter and result types in each function declaration are linked to their data type descriptions in Volume IV by page numbers written as comments.
- The **parameter descriptions** tell you what kind of information is passed into or out of the function.
- Constants that are passed are usually defined in one or more **constant descriptions** sections following the parameter descriptions. When a parameter description mentions a function or data type described elsewhere in this book, it cites the appropriate page number.
- The **discussion** tells you in detail what the function does and how you use it. Discussion sections often include code snippets that illustrate how and why you call the function from your code.
- Several sections of **other information** help you understand how to use the function in your programming. This information is labeled with these headings:
 - The **special considerations** section warns you about system factors, such as interrupts and prior calls, that may affect how the function works.
 - The **version notes** section summarizes the function's history in the QuickTime API and warn you of any QuickTime versions that do not support it. Functions introduced before QuickTime 3 are marked "Introduced in QuickTime 3 or earlier."
 - The **Programming info** section gives you the name of the header file where the function is declared and refers you to the Programming Summary section that discusses how the function is used.
 - If you are a Java programmer, the **related Java methods** section helps you understand how QuickTime for Java uses the function.
 - The **see also** section points you to further sources of information about the function and about related API elements in this and other books.

The chapters that document data structures, atoms, and callbacks follow a similar plan.

Volume I

Functions A–H

A

AbortPrePrerollMovie

Terminates the operation of PrePrerollMovie (II–1141).

```
void AbortPrePrerollMovie (
    Movie    m,
    OSErr    err );
```

m

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (II–1069), NewMovieFromFile (II–1080), and NewMovieFromHandle (II–1084).

err

See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

You normally call this function only if you have previously called PrePrerollMovie (II–1141) asynchronously and the user quits your application.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

Programming summary: “Enhancing Movie Playback Performance” (V–2722)

Related Java Methods

quicktime.std.movies.Movie.abortPrePreroll()

See Also

See PrePrerollMovie (II–1141).

AddCallBackToTimeBase

Places a **callback event** into the list of scheduled callback events.

```
OSErr AddCallBackToTimeBase (
    QTCallBack    cb );
```

AddClonedTrackToMovie

cb

Specifies the **callback event** for the operation. Your clock component obtains this value from the parameters passed to `ClockCallMeWhen` (I–91).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

If your component calls this function, the Movie Toolbox notifies it of time, rate, or stop and start changes via `ClockRateChanged` (I–97) and `ClockTimeChanged` (I–100).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Toolbox Clock Support Functions” (V–2869)

See Also

See `ClockNewCallBack` (I–96) and `ClockCallMeWhen` (I–91).

AddClonedTrackToMovie

Constructs a clone of an existing track in a movie.

```
OSErr AddClonedTrackToMovie (
    Track    sourceTrack,
    Movie    destinationMovie,
    long     flags,
    Track    *newTrack );
```

sourceTrack

Indicates the track to be cloned. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497). This is the source of the sample table once the cloned track is constructed.

destinationMovie

Indicates the movie where the cloned track should be created. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084). Currently, this must be the movie that contains the source track.

flags

Flags (see below) that determine how cloning should be performed. You currently must pass `kQTCOneShareSamples`.

newTrack

The address of storage where a reference to the newly constructed track is returned. If the function fails, this storage is set to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

flags Constants

`kQTCOneShareSamples`

Indicates that you want to share the sample table between the source and clone tracks. If you don’t pass this flag, currently, the `AddClonedTrackToMovie` call will fail.

`kQTCOneDontCopyEdits`

By default, the edit list is copied from the source track to the cloned track. Use this flag to leave the clone track’s edit list empty.

Special Considerations

Most QuickTime developers should never need to call this function.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `Movies.h`

See Also

This function is similar to `AddEmptyTrackToMovie` (I-23), in that it helps you create a new track and media compatible with a specified track.

AddEmptyTrackToMovie

Duplicates a track from a movie into the same movie or into another movie.

```
OSErr AddEmptyTrackToMovie (
    Track      srcTrack,
    Movie      dstMovie,
    Handle     dataRef,
    OSType     dataRefType,
    Track      *dstTrack );
```

AddFilePreview

`srcTrack`

The source track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

`dstMovie`

The destination movie for this operation. This can be the same movie as the source track or a different movie.

`dataRef`

A handle to the data reference. The type of information stored in the handle depends upon the data reference type specified by `dataRefType`.

`dataRefType`

The type of data reference; see “Data References” (IV–2661). If the data reference is an **alias**, you must set the parameter to `rAliasType`, indicating that the reference is an alias.

`dstTrack`

The newly created track’s identifier is returned in this parameter. If `AddEmptyTrackToMovie` fails, the resulting track identifier is set to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

This function returns a newly created, empty track. The newly created track has the same media type and track settings as the specified track. However, no data is copied from the source track to the new track. To copy data from the source track to the new track, use `InsertTrackSegment` (II–764) after calling `AddEmptyTrackToMovie`.

Version Notes

This function has been available since QuickTime 2.0.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Tracks” (V–2750)

Related Java Methods

`quicktime.std.movies.Movie.addEmptyTrack()`

AddFilePreview

Adds a **preview** to a file.

`OSErr AddFilePreview (`

```
short      resRefNum,
OSType     previewType,
Handle     previewData );
```

resRefNum

The **resource** file for this operation. You must have opened this resource file with write permission. If there is a **preview** in the specified file, the Movie Toolbox replaces that preview with a new one.

previewType

The **resource** type to be assigned to the **preview**. This type should correspond to the type of data stored in the preview. For example, if you have created a QuickDraw picture that you want to use as a preview for a file, you should set the previewType parameter to 'PICT'.

previewData

A handle to the **preview** data. For example, if the preview data is a picture, you would provide a picture handle.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Discussion

You must have created the **preview** data yourself. If the specified file already has a preview defined, the AddFilePreview function replaces it with the new preview.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Creating File Previews” (V-2757)

AddImageDescriptionExtension

Adds an extension to an ImageDescription (IV-2285) structure.

```
OSErr AddImageDescriptionExtension (
    ImageDescriptionHandle desc,
    Handle extension,
    long idType );
```

desc

A handle to the ImageDescription (IV-2285) structure to add the extension to.

extension

The handle containing the extension data.

AddMediaDataRef

idType

A four-byte signature identifying the type of data being added to the `ImageDescription`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function allows the application to add custom data to an `ImageDescriptionHandle`. This data could be specific to the compressor component referenced by the `ImageDescription` (IV–2285) structure.

Special Considerations

The Image Compression Manager makes a copy of the data referred to by the extension parameter. Thus, your application should dispose its copy of the data when it is no longer needed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Image Descriptions” (V–2790)

AddMediaDataRef

Adds a data reference to a media.

```
OSErr AddMediaDataRef (
    Media      theMedia,
    short      *index,
    Handle     dataRef,
    OSType     dataRefType );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

index

A pointer to a short integer. The Movie Toolbox returns the index value that is assigned to the new data reference. Your application can use this index to identify the reference to other Movie Toolbox functions, such as `GetMediaDataRef` (I–427). If the Movie Toolbox cannot add the data reference to the media, it sets the returned index value to 0.

dataRef

The data reference. This parameter contains a handle to the information that identifies the file that contains this media's data. The type of information stored in that handle depends upon the value of the *dataRefType* parameter.

dataRefType

The type of data reference. If the data reference is an **alias**, you must set this parameter to *rAliasType*.

function result You can access Movie Toolbox error returns through *GetMoviesError* (I-492) and *GetMoviesStickyError* (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: *Movies.h*

Programming summary: “Working With Data References” (V-2740)

Related Java Methods

quicktime.std.movies.media.Media.addDataRef()

See Also

See *Inside Macintosh: Files* (listed in the **bibliography**) for more information about **aliases** and the Alias Manager.

AddMediaSample

Adds sample data and a description to a media.

```
OSErr AddMediaSample (
    MediaHandle          theMedia,
    long                 dataIn,
    unsigned long         inOffset,
    TimeValue            size,
    SampleDescriptionHandle durationPerSample,
    long                 sampleDescriptionH,
    short                numberOfSamples,
    TimeValue            sampleFlags,
    *sampleTime );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as *NewTrackMedia* (II-1120) and *GetTrackMedia* (I-535).

AddMediaSample

`dataIn`

A handle to the sample data. The `AddMediaSample` function adds this data to the media specified by the parameter `theMedia`. You specify the number of bytes of sample data with the `size` parameter. You can use the `inOffset` parameter to specify a byte offset into the data referred to by this handle.

`inOffset`

Specifies an offset into the data referred to by the handle contained in the `dataIn` parameter. Set this parameter to 0 if there is no offset.

`size`

The number of bytes of sample data to be added to the media. This parameter indicates the total number of bytes in the sample data to be added to the media, not the number of bytes per sample. Use the `numberOfSamples` parameter to indicate the number of samples that are contained in the sample data.

`durationPerSample`

The duration of each sample to be added. You must specify this parameter in the media's time scale. For example, if you are adding sound that was sampled at 22 kHz to a media that contains a sound track with the same time scale, you would set `durationPerSample` to 1. Similarly, if you are adding video that was recorded at 10 frames per second to a video media that has a time scale of 600, you would set this parameter to 60 to add a single sample.

`sampleDescriptionH`

A handle to a `SampleDescription` (IV–2414) structure. Some media structures may require sample descriptions. There are different descriptions for different types of samples. For example, a media that contains compressed video requires that you supply an `ImageDescription` (IV–2285) structure. A media that contains sound requires that you supply a `SoundDescription` (IV–2449) structure. If the media does not require a `SampleDescription` structure, set this parameter to `NIL`.

`numberOfSamples`

The number of samples contained in the sample data to be added to the media. The Movie Toolbox considers the value of this parameter as well as the value of the `size` parameter when it determines the size of each sample that it adds to the media. You should set the value of this parameter so that the resulting sample size represents a reasonable compromise between total data retrieval time and the overhead associated with input and output (I/O). You should also consider the speed of the data storage device; CD-ROM devices are much slower than hard disks, for example, and should therefore have a smaller sample size. For a video media, set a sample size that corresponds to the size of a frame. For a sound media, choose a number of samples that corresponds to

between 0.5 and 1.0 seconds of sound. In general, you should not create groups of sound samples that are less than 2 KB in size or greater than 15 KB. Typically, a sample size of about 8 KB is reasonable for most storage devices.

`sampleFlags`

Contains flags (see below) that control the add operation. Set unused flags to 0.

`sampleTime`

A pointer to a time value. After adding the sample data to the media, the `AddMediaSample` function returns the time where the sample was inserted in the time value referred to by this parameter. If you don't want to receive this information, set this parameter to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

sampleFlags Constants

`mediaSampleNotSync`

Indicates that the sample to be added is not a **sync sample**. Set this flag to 1 if the sample is not a sync sample. Set this flag to 0 if the sample is a sync sample.

Discussion

Your application specifies the sample and the media for the operation. `AddMediaSample` updates the media so that it contains the sample data. One call to this function can add several samples to a media; however, all the samples must be the same size. Samples are always appended to the end of the media. Furthermore, the media duration is extended each time a sample is added.

```
// AddMediaSample coding example
// See “Discovering QuickTime,” page 250
#define kSoundSampleDuration    1
#define kSyncSample             0
#define kTrackStart             0
#define kMediaStart             0
#define kFix1                   0x00010000

void CreateMySoundTrack (Movie movie)
{
    Track                track;
    Media                media;
    Handle               hSound = NIL;
    SoundDescriptionHandle hSoundDesc = NIL;
    long                 lDataOffset;
```

AddMediaSample

```
long                lDataSize;
long                lNumSamples;

hSound = GetResource(soundListRsrc, 128);
if (hSound == NIL)
    return;

hSoundDesc = (SoundDescriptionHandle)NewHandle(4);

CreateMySoundDescription(hSound,
                        hSoundDesc,
                        &lDataOffset,
                        &lNumSamples,
                        &lDataSize);

track = NewMovieTrack(movie, 0, 0, kFullVolume);
media = NewTrackMedia(track, SoundMediaType,
                    FixRound((**hSoundDesc).sampleRate),
                    NIL, 0);

BeginMediaEdits(media);

AddMediaSample(media,
               hSound,
               lDataOffset,          // offset in data
               lDataSize,
               kSoundSampleDuration, // duration of each sound
                                   // sample
               (SampleDescriptionHandle)hSoundDesc,
               lNumSamples,
               kSyncSample,          // self-contained samples
               NIL);

EndMediaEdits(media);

InsertMediaIntoTrack(track,
                    kTrackStart,    // track start time
                    kMediaStart,    // media start time
                    GetMediaDuration(media),
                    kFix1);
```

```

        if (hSoundDesc != NIL)
            DisposeHandle((Handle)hSoundDesc);
    }

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Adding Samples to Media Structures” (V–2752)

Related Java Methods

`quicktime.std.movies.media.Media.addSample()`

See Also

See `AddMediaSampleReference` (I–31) and `AddMediaSampleReferences` (I–33).

AddMediaSampleReference

Works with samples that have already been added to a movie data file.

```

OSErr AddMediaSampleReference (
    Media                theMedia,
    long                dataOffset,
    unsigned long        size,
    TimeValue           durationPerSample,
    SampleDescriptionHandle sampleDescriptionH,
    long                numberOfSamples,
    short               sampleFlags,
    TimeValue           *sampleTime );

```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

dataOffset

The offset into the movie data file. This parameter is used differently by each data handler. For example, for the standard **HFS** data handler, this parameter specifies the offset into the file. This parameter contains either data you add yourself or the data offset returned by `GetMediaSampleReference` (I–447).

size

The number of bytes of sample data to be identified by the reference. This parameter indicates the total number of bytes in the sample data, not the

AddMediaSampleReference

number of bytes per sample. Use `numberOfSamples` to indicate the number of samples that are contained in the reference.

`durationPerSample`

The duration of each sample in the reference. You must specify this parameter in the media's time scale. For example, if you are referring to sound that was sampled at 22 kHz in a media that contains a sound track with the same time scale, to add a reference to a single sample you would set `durationPerSample` to 1. Similarly, if you are referring to video that was recorded at 10 frames per second in a video media that has a time scale of 60, you would set this parameter to 6 to add a reference to a single sample.

`sampleDescriptionH`

A handle to a `SampleDescription` (IV–2414) structure. Some media structures may require sample descriptions. There are different descriptions for different types of samples. For example, a media that contains compressed video requires that you supply an `ImageDescription` (IV–2285) structure. A media that contains sound requires that you supply a sound description structure. If the media does not require a `SampleDescription` structure, set this parameter to `NIL`.

`numberOfSamples`

The number of samples contained in the reference. For details, see `AddMediaSample` (I–27). If the media does not require a `SampleDescription` structure, set this parameter to `NIL`.

`sampleFlags`

Contains flags (see below) that control the operation. Set unused flags to 0.

`sampleTime`

A pointer to a time value. After adding the reference to the media, the `AddMediaSampleReference` function returns the time where the reference was inserted in the time value referred to by this parameter. If you don't want to receive this information, set this parameter to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

sampleFlags Constants

`mediaSampleNotSync`

Indicates that the sample to be added is not a **sync sample**. Set this flag to 1 if the sample is not a sync sample. Set this flag to 0 if the sample is a sync sample.

Discussion

This function does not add sample data to the file or device that contains a media. Rather, it defines references to sample data that you previously added to a movie

data file. Instead of actually writing out samples to disk, this function writes out references to existing samples, which you specify in `dataOffset` and the `size` parameter. As with `AddMediaSample` (I–27), your application specifies the media for the operation. Note that one reference may refer to more than one sample; all the samples described by a reference must be the same size. This function does not update the movie data file as part of the add operation. Therefore, your application does not have to call `BeginMediaEdits` (I–56) before calling `AddMediaSampleReference`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Adding Samples to Media Structures” (V–2752)

Related Java Methods

`quicktime.std.movies.media.Media.addSampleReference()`

See Also

See `AddMediaSample` (I–27) and `AddMediaSampleReferences` (I–33).

AddMediaSampleReferences

Adds groups of samples to a movie data file.

```
OSErr AddMediaSampleReferences (
    Media                theMedia,
    SampleDescriptionHandle sampleDescriptionH,
    long                 numberOfSamples,
    SampleReferencePtr    sampleRefs,
    TimeValue             *sampleTime );
```

`theMedia`

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

`sampleDescriptionH`

A handle to a `SampleDescription` (IV–2414) structure. Some media structures may require sample descriptions. There are different descriptions for different types of samples. For example, a media that contains compressed video requires that you supply an `ImageDescription` (IV–2285) structure. A media that contains sound requires that you supply a sound description structure. If you don't want the `SampleDescription` structure, set this parameter to `NIL`.

AddMediaSampleReferences64

`numberOfSamples`

The number of samples contained in the reference. For details, see `AddMediaSample` (I–27).

`sampleRefs`

A pointer to the number of `SampleReferenceRecord` structures specified in the `numberOfSamples` parameter.

`sampleTime`

A pointer to a time value. After adding the reference to the media, the `AddMediaSampleReferences` function returns the time where the reference was inserted, using the time scale referred to by this parameter. If you don't want to receive this information, set this parameter to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Using this function instead of `AddMediaSampleReference` (I–31) can greatly improve the performance of operations that involve adding a large number of samples to a movie at one time. `AddMediaSampleReferences` provides no capabilities that weren't previously available with `AddMediaSampleReference`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Adding Samples to Media Structures” (V–2752)

Related Java Methods

`quicktime.std.movies.media.Media.addSampleReference()`

See Also

See `AddMediaSample` (I–27) and `AddMediaSampleReference` (I–31).

AddMediaSampleReferences64

Provides a 64-bit version of `AddMediaSampleReferences` (I–33).

```
OSErr AddMediaSampleReferences64 (
    Media                theMedia,
    SampleDescriptionHandle sampleDescriptionH,
    long                 numberOfSamples,
    SampleReference64Ptr sampleRefs,
```



```
TimeValue *sampleTime );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

sampleDescriptionH

A handle to a `SampleDescription` (IV–2414) structure. Some media structures may require sample descriptions. There are different descriptions for different types of samples. For example, a media that contains compressed video requires that you supply an `ImageDescription` (IV–2285) structure. A media that contains sound requires that you supply a sound description structure. If you don't want the `SampleDescription` structure, set this parameter to `NIL`.

numberOfSamples

The number of samples contained in the reference. For details, see `AddMediaSample` (I–27).

sampleRefs

A pointer to the number of `SampleReference64Record` structures specified in the `numberOfSamples` parameter.

sampleTime

A pointer to a time value. After adding the reference to the media, the `AddMediaSampleReferences` function returns the time where the reference was inserted, using the time scale referred to by this parameter. If you don't want to receive this information, set this parameter to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The only difference between this function and `AddMediaSampleReferences` (I–33) is that the `sampleRefs` parameter points to `SampleReference64Record` structures instead of `SampleReferenceRecord` structures.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

AddMovieExecuteWiredActionsProc

See Also

See AddMediaSampleReferences (I–33).

AddMovieExecuteWiredActionsProc

Lets you add a callback to a movie to execute wired actions.

```
OSErr AddMovieExecuteWiredActionsProc (
    Movie          theMovie,
    MovieExecuteWiredActionsUPP  proc,
    void           *refCon );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as NewMovie (II–1069), NewMovieFromFile (II–1080), and NewMovieFromHandle (II–1084).

proc

A callback function, as described in MovieExecuteWiredActionsProc (III–2101).

refCon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

AddMovieResource

Adds a movie **resource** to a specified resource file.

```
OSErr AddMovieResource (
    Movie          theMovie,
    short          resRefNum,
    short          *resId,
    ConstStr255Param  resName );
```

`theMovie`

The movie you wish to add to the movie file. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

`resRefNum`

Identifies the movie file to which the **resource** is to be added. Your application obtains this value from the `OpenMovieFile` (II–1133) function.

`resId`

A pointer to a field that contains the **resource** ID number for the new resource. If the field referred to by `resId` is set to 0, the Movie Toolbox assigns a unique resource ID number to the new resource. The toolbox then returns the movie's resource ID number in the field referred to by the `resId` parameter.

`AddMovieResource` assigns resource ID numbers sequentially, starting at 128. If `resId` is set to `NIL`, the Movie Toolbox assigns a unique resource ID number to the new resource and does not return that resource's ID value. Set `resId` to `movieInDataForkResID` to add the new resource to the movie file's data fork (see below).

`resName`

Points to a character string that contains the name of the movie **resource**. If you set `resName` to `NIL`, the toolbox creates an unnamed resource.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

resId Constants

`movieInDataForkResID`

Adds a **resource** to a movie file's data fork.

Discussion

This function adds the movie to the file, effectively saving any changes you have made to the movie. To use this function with single-fork movie files, pass `movieInDataForkResID` as the `resId` parameter. After updating the movie file, `AddMovieResource` clears the movie changed flag, indicating that the movie has not been changed.

```
// AddMovieResource coding example
// See “Discovering QuickTime,” page 243
void CreateMyCoolMovie (void)
{
    StandardFileReply  sfr;
    Movie              movie = NIL;
```

AddMovieSelection

```
FSSpec          fss;
short           nFileRefNum = 0;
short           nResID = movieInDataForkResID;

StandardPutFile("\pEnter movie file name:", "\puntitled.mov", &sfr);
if (!sfr.sfGood)
    return;

CreateMovieFile(&sfr.sfFile,
               FOUR_CHAR_CODE('TVOD'),
               smCurrentScript,
               createMovieFileDeleteCurFile |
               createMovieFileDontCreateResFile,
               &nFileRefNum,
               &movie);

CreateMyVideoTrack(movie);      // See next section
CreateMySoundTrack(movie);      // See next section

AddMovieResource(movie, nFileRefNum, &nResID, NIL);
if (nFileRefNum != 0)
    CloseMovieFile(nFileRefNum);
DisposeMovie(movie);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Saving Movies” (V-2715)

Related Java Methods

`quicktime.std.movies.Movie.addResource()`

AddMovieSelection

Adds one or more tracks to a movie.

```
void AddMovieSelection (
    Movie    theMovie,
    Movie    src );
```

theMovie

The destination movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (II-1069), NewMovieFromFile (II-1080), and NewMovieFromHandle (II-1084).

src

The source movie for this operation. AddMovieSelection adds the tracks from this movie to the destination movie. The function adds these tracks at the time specified by the current selection in the destination movie.

function result You can access this function's error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494).

Discussion

This function scales the source movie so that it fits into the destination selection. If the current selection in the destination movie has a 0 duration, the Movie Toolbox adds the segment at the beginning of the current selection. The entire source movie is used regardless of the selection in the source movie. The Movie Toolbox removes any empty tracks from the destination movie after the add operation. If you have assigned a **progress function** to the destination movie, the Movie Toolbox calls that progress function during long add operations. Following is an example of using this function:

```
// AddMovieSelection coding example
// See "Discovering QuickTime," page 363
Movie          movie1;
TimeValue      101dDuration;
Movie          movie2;
long           1Index, 1OrigTrackCount, 1ReferenceIndex;
Track          track, trackSprite;

// get the first track in original movie and position at the start
trackSprite = GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);

// remove all tracks except video in modifier movie
for (1Index = 1; 1Index <= GetMovieTrackCount(movie2); 1Index++) {
    Track          track = GetMovieIndTrack(movie2, 1Index);
    OSType          dwType;

    GetMediaHandlerDescription(GetTrackMedia(track),
                              &dwType, NIL, NIL);
    if (dwType != VideoMediaType) {
```

AddSoundDescriptionExtension

```
        DisposeMovieTrack(track);
        lIndex--;
    }
}

// add the modifier track to original movie
lOldDuration = GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);

// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, lOldDuration,
                    GetMovieDuration(movie1) - lOldDuration);

// associate the modifier track with the original sprite track
track = GetMovieIndTrack(movie1, lOrigTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                  &lReferenceIndex);
```

Special Considerations

Some Movie Toolbox functions can take a long time to execute. For example, if you call `FlattenMovie` (I–372) and specify a large movie, the Movie Toolbox must read and write all the sample data for the movie. During such operations you may wish to display some kind of progress indicator to the user. A **progress function** is an application-defined function that you can create to track the progress of time-consuming activities and keep the user informed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Movie Editing Functions” (V–2746)

Related Java Methods

`quicktime.std.movies.Movie.addSelection()`

AddSoundDescriptionExtension

Adds an extension to a `SoundDescription` (IV–2449) structure.

```
OSErr AddSoundDescriptionExtension (
    SoundDescriptionHandle desc,
```

```

    Handle          extension,
    OSType          idType );

```

desc

A handle to the `SoundDescription` (IV–2449) structure to add the extension to.

extension

The handle containing the extension data.

idType

A four-byte signature identifying the type of data being added to the `SoundDescription`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Two extensions are defined to the `SoundDescription` record. The first is the slope, intercept, `minClip`, and `maxClip` parameters for audio, represented as an atom of type ‘flap’ (IV–2537). The second extension is the ability to store data specific to a given audio codec, using a `SoundDescriptionV1` (IV–2451) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.media.SoundDescription.addExtension()`

AddTime

Adds two times.

```

void AddTime (
    TimeRecord    *dst,
    const TimeRecord *src );

```

dst

A pointer to a **time structure**. This time structure contains one of the operands for the addition. `AddTime` returns the result of the addition into this time structure.

AddTrackReference

src

A pointer to a **time structure**. The Movie Toolbox adds this value to the time or duration specified by the *dst* parameter.

function result You can access this function's error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Discussion

You must specify the input times in **time structures**. The result value is formatted as a duration or a time value, the same as the format of the structure pointed to by the *dst* parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Working With Times" (V-2762)

Related Java Methods

`quicktime.std.clocks.TimeRecord.addTime()`

AddTrackReference

Adds a new track reference to a track.

```
OSErr AddTrackReference (
    Track    theTrack,
    Track    refTrack,
    OSType   refType,
    long     *addedIndex );
```

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II-1092) and `GetMovieTrack` (I-497).

refTrack

The track to be identified in the track reference.

refType

The type of reference.

addedIndex

A pointer to a long integer. The toolbox returns the index value assigned to the new track reference. If you don't want this information, set this parameter to NIL.

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

The following code snippet shows how AddTrackReference can be used to add a modifier track reference to a **sprite** track.

```
// AddTrackReference coding example
// See “Discovering QuickTime,” page 363
Movie          movie1;
TimeValue      101dDuration;
Movie          movie2;
long           1Index, 1OrigTrackCount, 1ReferenceIndex;
Track          track, trackSprite;

// get the first track in original movie and position at the start
trackSprite = GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);

// remove all tracks except video in modifier movie
for (1Index = 1; 1Index <= GetMovieTrackCount(movie2); 1Index++) {
    Track          track = GetMovieIndTrack(movie2, 1Index);
    OSType          dwType;

    GetMediaHandlerDescription(GetTrackMedia(track),
                              &dwType, NIL, NIL);
    if (dwType != VideoMediaType) {
        DisposeMovieTrack(track);
        1Index--;
    }
}

// add the modifier track to original movie
101dDuration = GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);
```

AddUserData

```
// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, 10ldDuration,
                    GetMovieDuration(movie1) - 10ldDuration);

// associate the modifier track with the original sprite track
track = GetMovieIndTrack(movie1, 10origTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                  &lReferenceIndex);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Track References” (V–2737)

Related Java Methods

`quicktime.std.movies.Track.addReference()`

AddUserData

Adds an item to a **user data list**.

```
OSErr AddUserData (
    UserData    theUserData,
    Handle      data,
    OSType      udType );
```

theUserData

The **user data list** for this operation. You obtain this item reference by calling `GetMovieUserData` (I–499), `GetTrackUserData` (I–546), or `GetMediaUserData` (I–456).

data

A handle to the data to be added to the **user data list**.

udType

The type that is to be assigned to the new item.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You specify the **user data list**, the data to be added, and the data's type value.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

`quicktime.std.movies.media.UserData.addData()`

AddUserDataText

Places language-tagged text into an item in a **user data list**.

```
OSErr AddUserDataText (
    UserData    theUserData,
    Handle      data,
    OSType      udType,
    long        index,
    short       itlRegionTag );
```

theUserData

The **user data list** for this operation. You obtain this list reference by calling `GetMovieUserData` (I–499), `GetTrackUserData` (I–546), or `GetMediaUserData` (I–456).

data

A handle to the data to be added to the **user data list**.

udType

The type that is to be assigned to the new item.

index

The item to which the text is to be added. This parameter must specify an item in the **user data list** identified by *theUserData*.

itlRegionTag

The **region code** of the text to be added. If there is already text with this region code in the item, the function replaces the existing text with the data specified by the *data* parameter. See *Inside Macintosh: Text* (listed in the **bibliography**) for more information about language and region codes.

function result You can access Movie Toolbox error returns through `GetMoviesError`

AlignScreenRect

(I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You specify the **user data list** and item, the data to be added, the data’s type value, and the language code of the data.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

`quicktime.std.movies.media.UserData.addText()`

AlignScreenRect

Aligns a specified rectangle to the strictest screen that the rectangle intersects.

```
void AlignScreenRect (
    Rect          *rp,
    ICMAAlignmentProcRecordPtr alignmentProc );
```

`rp`

A pointer to a rectangle defined in global screen coordinates.

`alignmentProc`

Points to your own alignment behavior function. Set this parameter to `NIL` to use the standard behavior.

Discussion

For a specification of your alignment function, see `ICMAAlignmentProc` (III–2086).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Aligning Windows” (V–2797)

See Also

The callback function `MyICMAAlignmentProc` is described in `ICMAAlignmentProc`.

AlignWindow

Moves a specified window to the nearest optimal alignment position.

```
void AlignWindow (
    WindowPtr          wp,
    Boolean             front,
    const Rect         *alignmentRect,
    ICMAAlignmentProcRecordPtr alignmentProc );
```

wp

Points to the window to be aligned.

front

The frontmost window. If the `front` parameter is `TRUE` and the window specified in the `wp` parameter isn't the active window, `AlignWindow` makes it the active window.

alignmentRect

A pointer to a rectangle in window coordinates that allows you to align the window to a rectangle within the window. Set this parameter to `NIL` to align using the bounds of the window.

alignmentProc

Points to a function that allows you to provide your own alignment behavior. Set this parameter to `NIL` to use the standard behavior.

Discussion

For a specification of your alignment function, see `ICMAAlignmentProc` (III–2086).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Aligning Windows” (V–2797)

See Also

The callback function `MyICMAAlignmentProc` is described in `ICMAAlignmentProc`.

AudioGetBass

Deprecated.

```
ComponentResult AudioGetBass (
    ComponentInstance ac,
```

AudioGetInfo

```
short          whichChannel,  
short          *bass );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

AudioGetInfo

Deprecated.

```
ComponentResult AudioGetInfo (  
    ComponentInstance  ac,  
    AudioInfoPtr       info );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

AudioGetMute

Deprecated.

```
ComponentResult AudioGetMute (  
    ComponentInstance  ac,  
    short              whichChannel,  
    short              *mute );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

AudioGetOutputDevice

Deprecated.

```
ComponentResult AudioGetOutputDevice (
    ComponentInstance  ac,
    Component          *outputDevice );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

AudioGetTreble

Deprecated.

```
ComponentResult AudioGetTreble (
    ComponentInstance  ac,
    short              whichChannel,
    short              *Treble );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

AudioGetVolume

Deprecated.

```
ComponentResult AudioGetVolume (
    ComponentInstance  ac,
    short              whichChannel,
    ShortFixed         *volume );
```

AudioMuteOnEvent

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

AudioMuteOnEvent

Deprecated.

```
ComponentResult AudioMuteOnEvent (
    ComponentInstance  ac,
    short              muteOnEvent );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

AudioSetBass

Deprecated.

```
ComponentResult AudioSetBass (
    ComponentInstance  ac,
    short              whichChannel,
    short              bass );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

AudioSetMute

Deprecated.

```
ComponentResult AudioSetMute (
    ComponentInstance  ac,
    short              whichChannel,
    short              mute );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

AudioSetToDefaults

Deprecated.

```
ComponentResult AudioSetToDefaults (
    ComponentInstance  ac );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

AudioSetTreble

Deprecated.

```
ComponentResult AudioSetTreble (
    ComponentInstance  ac,
    short              whichChannel,
    short              Treble );
```

AudioSetVolume

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

AudioSetVolume

Deprecated.

```
ComponentResult AudioSetVolume (
    ComponentInstance  ac,
    short              whichChannel,
    ShortFixed         volume );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime. Macintosh Note: Not available in **CarbonLib**, but available when SoundLib 3.0 or later is installed.

Programming Info

C interface file: Sound.h

B

BeginFullScreen

Begins full-screen mode for a specified monitor.

```
OSErr BeginFullScreen (
    Ptr          *restoreState,
    GDHandle     whichGD,
    short        *desiredWidth,
    short        *desiredHeight,
    WindowPtr    *newWindow,
    RGBColor     *eraseColor,
    long         flags );
```

`restoreState`

On exit, a pointer to a block of private state data that contains information on how to return from full-screen mode. This value is passed to `EndFullScreen` (I-314) to enable it to return the monitor to its previous state.

`whichGD`

A handle to the graphics device to put into full-screen mode. Set this parameter to `NIL` to select the main screen.

`desiredWidth`

On entry, a pointer to a short integer that contains the desired width, in pixels, of the images to be displayed. On exit, that short integer is set to the actual number of pixels that can be displayed horizontally. Set this parameter to 0 to leave the width of the display unchanged.

`desiredHeight`

On entry, a pointer to a short integer that contains the desired height, in pixels, of the images to be displayed. On exit, that short integer is set to the actual number of pixels that can be displayed vertically. Set this parameter to 0 to leave the height of the display unchanged.

`newWindow`

On entry, a window-creation value. If this parameter is `NIL`, no window is created for you. If this parameter has any other value, `BeginFullScreen` creates a new window that is large enough to fill the entire screen and returns a pointer to that window in this parameter. You should not dispose of that window yourself; instead, `EndFullScreen` (I-314) will do so.

`eraseColor`

The color to use when erasing the full-screen window created by `BeginFullScreen` if `newWindow` is not `NIL` on entry. If this parameter is `NIL`, `BeginFullScreen` uses black when initially erasing the window's content area.

`flags`

A set of bit flags (see below) that control certain aspects of the full-screen mode.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

flags Constants

`fullScreenHideCursor`

If this flag is set, `BeginFullScreen` hides the cursor. This is useful if you are going to play a QuickTime movie and don't want the cursor to be visible over the movie.

BeginFullScreen

fullScreenAllowEvents

If this flag is set, your application intends to allow other applications to run (with Macintosh, by calling `WaitNextEvent` to grant them processing time). In this case, `BeginFullScreen` does not change the monitor resolution, because other applications might depend on the current resolution.

fullScreenDontChangeMenuBar

If this flag is set, `BeginFullScreen` does not hide the menu bar. This is useful if you want to change the resolution of the monitor but still need to allow the user to access the menu bar.

fullScreenPreflightSize

If this flag is set, `BeginFullScreen` doesn't change any monitor settings, but returns the actual height and width that it would use if this bit were not set. This allows applications to test for the availability of a monitor setting without having to switch to it.

Discussion

This function returns, in the `restoreState` parameter, a pointer to a block of private state information that indicates how to return from full-screen mode. You pass that pointer as a parameter to the `EndFullScreen` (I-314) function. The following sample code contains functions that illustrate how to play a QuickTime movie full screen. It prompts the user for a movie, opens that movie, configures it to play full screen, associates a movie controller, and lets the controller handle events. Your application would call `QTFullScreen_EventLoopAction` in its event loop (on the Mac OS) or when it gets idle events (on Windows).

```
enum {
    fullScreenHideCursor          = 1L << 0,
    fullScreenAllowEvents         = 1L << 1,
    fullScreenDontChangeMenuBar   = 1L << 2,
    fullScreenPreflightSize       = 1L << 3
};

// QTFullScreen_PlayOnFullScreen
// Prompt the user for a movie and play it full screen.
OSErr QTFullScreen_PlayOnFullScreen (void)
{
    FSSpec          myFSSpec;
    Movie           myMovie = NIL;
    short           myRefNum = 0;
    SFTYPEList      myTypeList = {MovieFileType, 0, 0, 0};
    StandardFileReply myReply;
```

```

long                myFlags = fullScreenDontChangeMenuBar
                                | fullScreenAllowEvents;
OSErr                myErr = noErr;

StandardGetFilePreview(NIL, 1, myTypeList, &myReply);
if (!myReply.sfGood)
    goto bail;

// make an FSSpec record
FSMakeFSSpec(myReply.sfFile.vRefNum, myReply.sfFile.parID,
                                myReply.sfFile.name, &myFSSpec);

myErr = OpenMovieFile(&myFSSpec, &myRefNum, fsRdPerm);
if (myErr != noErr)
    goto bail;

// now fetch the first movie from the file
myErr = NewMovieFromFile(&myMovie, myRefNum, NIL, NIL,
                                newMovieActive, NIL);

if (myErr != noErr)
    goto bail;

CloseMovieFile(myRefNum);

// set up for full-screen display
myErr = BeginFullScreen(&gRestoreState, NIL, 0, 0,
                                &gFullScreenWindow, NIL, myFlags);

#ifdef TARGET_OS_WIN32
    // on Windows, set a window procedure for the new window
    // and associate a port with that window
    QTMLSetWindowWndProc(gFullScreenWindow, QTFullScreen_HandleMessages);
    CreatePortAssociation(GetPortNativeWindow(gFullScreenWindow), NIL, 0L);
#endif

SetMovieGWorld(myMovie, (CGrafPtr)gFullScreenWindow,
                                GetGWorldDevice((CGrafPtr)gFullScreenWindow));
SetMovieBox(myMovie, &gFullScreenWindow->portRect);

// create the movie controller
gMC = NewMovieController(myMovie, &gFullScreenWindow->portRect, 0);

```

BeginMediaEdits

Version Notes

The Macintosh human interface guidelines suggest that the menu bar must always be present, and that information must always appear in windows. However, many multimedia applications have chosen to change the look and feel of the interface based on their needs. The number of details to keep track of when doing this continues to increase. To help solve this problem, QuickTime 2.1 added functions to put a graphics device into full screen mode. The key elements to displaying full screen movies are the calls `BeginFullScreen` and `EndFullScreen`, introduced in QuickTime 2.5.

Programming Info

C interface file: `Movies.h`

Programming summary: “Using the Full Screen” (V–2717)

Related Java Methods

```
quicktime.std.movies.FullScreen.getMainScreenSize(),
quicktime.std.movies.FullScreen.getScreenSize(),
quicktime.std.movies.FullScreen.preflightSize(),
quicktime.std.movies.FullScreen.begin()
```

See Also

See `EndFullScreen` (I–314).

BeginMediaEdits

Starts a media-editing session.

```
OSErr BeginMediaEdits (
    Media    theMedia );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Use `EndMediaEdits` (I–335) to end a media-editing session. You must call `BeginMediaEdits` before you add samples to a media with the `AddMediaSample` (I–27) function. You must also call `BeginMediaEdits` before calling `InsertTrackSegment` (II–764) if you wish `InsertTrackSegment` to copy media samples instead of copying the segment by reference.

```

// BeginMediaEdits coding example
// See "Discovering QuickTime," page 89
void CreateMyVideoTrack (Movie movie)
{
    Track    track;
    Media    media;
    Rect     rect = {0, 0, 100, 320};

    track = NewMovieTrack(movie,
                          FixRatio(rect.right, 1),
                          FixRatio(rect.bottom, 1),
                          kNoVolume);

    media = NewTrackMedia(track,
                          VideoMediaType,
                          600,                      // video time scale
                          NIL, NIL);

    BeginMediaEdits(media);
    MyAddVideoSamplesToMedia(media, &rect);    // assemble data
    EndMediaEdits(media);
    InsertMediaIntoTrack(track,
                          0,                      // track start time
                          0,                      // media start time
                          GetMediaDuration(media),
                          kFix1);                // normal speed
}

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Adding Samples to Media Structures" (V-2752)

Related Java Methods

`quicktime.std.movies.media.Media.beginEdits()`

See Also

See `InsertTrackSegment` (II-764).

CallComponent

C

CallComponent

Invokes a specified function of your component with the parameters originally provided by the application that called your component.

```
ComponentResult CallComponent (
    ComponentInstance    ci,
    ComponentParameters   *cp );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

cp

A pointer to the `ComponentParameters` (IV–2216) record that your component received from the Component Manager.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

CallComponentCanDo

Calls a component directly to query its capabilities.

```
ComponentResult CallComponentCanDo (
    ComponentInstance    ci,
    short                ftnNumber );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

ftnNumber

A request code value. See *Inside Macintosh: QuickTime Components* (listed in the **bibliography**) for information about the request codes supported by the

QuickTime components supplied by Apple. For other components, see their developer's documentation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

CallComponentClose

Calls a component directly to terminate your application's access to it.

```
ComponentResult CallComponentClose (
    ComponentInstance ci,
    ComponentInstance self );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

self

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

CallComponentDispatch

Replaces `CallComponent` (I–58) for Mac OS 9 and later.

```
ComponentResult CallComponentDispatch (
    ComponentParameters *cp );
```

cp

A pointer to the `ComponentParameters` (IV–2216) record that your component received from the Component Manager.

CallComponentExecuteWiredAction

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Components.h

CallComponentExecuteWiredAction

Undocumented

```
ComponentResult CallComponentExecuteWiredAction (
    ComponentInstance      ci,
    QTAtomContainer        actionContainer,
    QTAtom                  actionAtom,
    QTCustomActionTargetPtr target,
    QTEventRecordPtr        event );
```

ci

A component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

actionContainer

A QT atom container that contains the action atom.

actionAtom

The action atom for this wired action.

target

A pointer to a QTCustomActionTargetRecord (IV–2351) structure.

event

A pointer to a QTEventRecord (IV–2353) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: MediaHandlers.h

CallComponentFunction

Invokes a specified function of your component with the parameters originally provided by the application that called your component.

```
long CallComponentFunction (
    ComponentParameters    *params,
    ComponentFunctionUPP    func );
```

params

A pointer to the `ComponentParameters` (IV–2216) record that your component received from the Component Manager.

func

The address of the function that is to handle the request. The Component Manager calls the routine referred to by the `func` parameter as a Pascal function with the parameters that were originally provided by the application. These parameters are preceded by a handle to the memory associated with the current connection. The routine referred to by the `func` parameter must return a function result of type `ComponentResult` (a long integer) indicating the success or the failure of the operation.

function result A `ComponentResult` value that indicates the success or the failure of the operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

CallComponentFunctionWithStorage

Invokes a specified function of your component with the parameters originally provided by the application that called your component and provides a handle to the memory associated with the current connection.

```
long CallComponentFunctionWithStorage (
    Handle                storage,
    ComponentParameters    *params,
    ComponentFunctionUPP    func );
```

CallComponentFunctionWithStorageProcInfo

storage

A handle to the memory associated with the current connection. The Component Manager provides this handle to your component along with the request.

params

A pointer to the ComponentParameters (IV–2216) record that your component received from the Component Manager.

func

The address of the function that is to handle the request. The Component Manager calls the routine referred to by the *func* parameter as a Pascal function with the parameters that were originally provided by the application. These parameters are preceded by a handle to the memory associated with the current connection. The routine referred to by the *func* parameter must return a function result of type *ComponentResult* (a long integer) indicating the success or the failure of the operation.

function result A *ComponentResult* value that indicates the success or the failure of the operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: *Components.h*

Programming summary: “Functions Used By Components” (V–2782)

CallComponentFunctionWithStorageProcInfo

Invokes a specified function of your component with the parameters originally provided by the application that called it; used when your component links with *ComponentsInterfacesLib*.

```
long CallComponentFunctionWithStorageProcInfo (  
    Handle                storage,  
    ComponentParameters   *params,  
    ProcPtr               func,  
    ProcInfoType          funcProcInfo );
```

storage

A handle to the memory associated with the current connection. The Component Manager provides this handle to your component along with the request.

params

A pointer to the ComponentParameters (IV-2216) record that your component received from the Component Manager.

func

The address of the function that is to handle the request. The Component Manager calls the routine referred to by the func parameter as a Pascal function with the parameters that were originally provided by the application. These parameters are preceded by a handle to the memory associated with the current connection. The routine referred to by the func parameter must return a function result of type ComponentResult (a long integer) indicating the success or the failure of the operation. Note that for PowerPC code, the func parameter should still point to the routine itself, not to a RoutineDescriptor or **Universal Procedure Pointer**.

funcProcInfo

The procedure information for the routine referred to by the func parameter. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**) for information on procedure information data.

function result A ComponentResult value that indicates the success or the failure of the operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Components.h

Programming summary: “Functions Used By Components” (V-2782)

CallComponentGetMPWorkFunction

Undocumented

```
ComponentResult CallComponentGetMPWorkFunction (
    ComponentInstance      ci,
    ComponentMPWorkFunctionUPP *workFunction,
    void                  **refCon );
```

ci

A component instance. Your software obtains this reference from OpenComponent (II-1130) or OpenDefaultComponent (II-1131).

CallComponentGetPublicResource

workFunction

A `ComponentMPWorkFunctionProc` (III–2077) callback.

refCon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

CallComponentGetPublicResource

Undocumented

```
ComponentResult CallComponentGetPublicResource (
    ComponentInstance ci,
    OSType             resourceType,
    short              resourceID,
    Handle              *resource );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

resourceType

Undocumented

resourceID

A **resource** ID.

resource

On return, pointer to a handle to the **resource**.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Components.h`

CallComponentOpen

Calls a component directly to let an application gain access to its services.

```
ComponentResult CallComponentOpen (
    ComponentInstance    ci,
    ComponentInstance    self );
```

ci

A component instance. Your software obtains this reference from
OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

self

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Components.h

CallComponentRegister

Calls a component directly to register it.

```
ComponentResult CallComponentRegister (
    ComponentInstance    ci );
```

ci

A component instance. Your software obtains this reference from
OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Components.h

CallComponentTarget

CallComponentTarget

Calls a component directly to set its target.

```
ComponentResult CallComponentTarget (  
    ComponentInstance    ci,  
    ComponentInstance    target );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

target

The component instance of the component issuing the target request.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

CallComponentUnregister

Calls a component directly to unregister it.

```
ComponentResult CallComponentUnregister (  
    ComponentInstance    ci );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

CallComponentVersion

Calls a component directly to retrieve its version number.

```
ComponentResult CallComponentVersion (
    ComponentInstance ci );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

CallMeWhen

Schedules a **callback event**.

```
OSErr CallMeWhen (
    QTCallback cb,
    QTCallbackUPP callBackProc,
    long refCon,
    long param1,
    long param2,
    long param3 );
```

cb

The **callback event** for the operation. You obtain this identifier from `NewCallback` (II–1053).

callBackProc

Points to your callback function, described in `QTCallbackProc` (III–2124).

refCon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

param1

Contains scheduling information. The Movie Toolbox interprets this parameter based on the value of the `cbType` parameter to `NewCallback` (II–1053). If `cbType` is

CallMeWhen

set to `callBackAtTime`, the `param1` parameter contains flags (see below) indicating when to invoke your callback function for this **callback event**. If the `cbType` parameter is set to `callBackAtRate`, `param1` contains flags (see below) indicating when to invoke your callback function for this event. Be sure to set unused flags to 0.

`param2`

Contains scheduling information. The Movie Toolbox interprets this parameter based on the value of the `cbType` parameter to `NewCallBack` (II–1053). If `cbType` is set to `callBackAtTime`, the `param2` parameter contains the time value at which your callback function is to be invoked for this event. The `param1` parameter contains flags affecting when the Movie Toolbox calls your function. If `cbType` is set to `callBackAtRate`, the `param2` parameter contains the rate value at which your callback function is to be invoked for this event. The `param1` parameter contains flags affecting when the Movie Toolbox calls your function.

`param3`

The time scale in which to interpret the time value that is stored in `param3` if `cbType` is set to `callBackAtTime`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

callBackAtTime Constants

`triggerTimeFwd`

Indicates that your callback function should be called at the time specified by `param2` only when time is moving forward (positive rate). The value of this flag is 0x0001.

`triggerTimeBwd`

Indicates that your callback function should be called at the time specified by `param2` only when time is moving backward (negative rate). The value of this flag is 0x0002.

`triggerTimeEither`

Indicates that your callback function should be called at the time specified by `param2` without regard to direction, but the rate must be nonzero. The value of this flag is 0x0003.

callBackAtRate Constants

`triggerRateChange`

Indicates that your callback function should be called whenever the rate changes. The value of this flag is 0x0000.

`triggerRateLT`

Indicates that your callback function should be called when the rate changes to a value less than that specified by `param2`. The value of this flag is 0x0004.

`triggerRateGT`

Indicates that your callback function should be called when the rate changes to a value greater than that specified by `param2`. The value of this flag is 0x0008.

`triggerRateEqual`

Indicates that your callback function should be called when the rate changes to a value equal to that specified by `param2`. The value of this flag is 0x0010.

`triggerRateLTE`

Indicates that your callback function should be called when the rate changes to a value that is less than or equal to that specified by `param2`. The value of this flag is 0x0014.

`triggerRateGTE`

Indicates that your callback function should be called when the rate changes to a value that is greater than or equal to that specified by `param2`. The value of this flag is 0x0018.

`triggerRateNotEqual`

Indicates that your callback function should be called when the rate changes to a value that is not equal to that specified by `param2`. The value of this flag is 0x001C.

Discussion

You can call this function from your callback function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Time Base Callback Functions” (V–2763)

Related Java Methods

```
quicktime.std.clocks.RateCallback.callMeWhen(),
quicktime.std.clocks.TimeCallback.callMeWhen(),
quicktime.std.clocks.QTCallback.callMeWhen(),
quicktime.std.clocks.TimeJumpCallback.callMeWhen(),
quicktime.std.clocks.ExtremesCallback.callMeWhen()
```

See Also

For a specification of your callback function, see `QTCallbackProc` (III–2124).

CancelCallback

CancelCallback

Cancels a **callback event** before it executes.

```
void CancelCallback (
    QTCallback    cb );
```

cb

The **callback event** for this operation. You obtain this value from NewCallback (II–1053).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Time Base Callback Functions” (V–2763)

Related Java Methods

quicktime.std.clocks.QTCallback.cancel()

See Also

See NewCallback (II–1053).

CanQuickTimeOpenDataRef

Determines whether referenced data can be opened using a graphics importer or opened in place as a movie.

```
OSErr CanQuickTimeOpenDataRef (
    Handle    dataRef,
    OSType    dataRefType,
    Boolean    *outCanOpenWithGraphicsImporter,
    Boolean    *outCanOpenAsMovie,
    Boolean    *outPreferGraphicsImporter,
    UInt32    inFlags );
```

dataRef

A handle to the referenced data.

dataRefType

The type of data reference pointed to by dataRef; see “Data References” (IV–2661).

outCanOpenWithGraphicsImporter

Points to a Boolean that will be set to TRUE if the file can be opened using a graphics importer and FALSE otherwise. If you do not want this information, pass NIL.

outCanOpenAsMovie

Points to a Boolean that will be set to TRUE if the file can be opened as a movie and FALSE otherwise. If you do not want this information, pass NIL.

outPreferGraphicsImporter

Points to a boolean which will be set to true if the file can be opened using a graphics importer and opened as a movie, but, other factors being equal, QuickTime prefers a graphics importer. For example, QuickTime recommends using a graphics importer for single-frame GIF files and opening as a movie for multiple-frame GIF files. If you do not want this information, pass NIL. Passing a valid pointer disables the kQTDontUseDataToFindImporter and kQTDontLookForMovieImporterIfGraphicsImporterFound flags, if set.

inFlags

Flags (see below) that modify search behavior. Pass 0 for default behavior.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

inFlags Constants

kQTDontUseDataToFindImporter

Tells QuickTime not to use the data in the file to help in the search. This will speed up the search, especially in cases where a negative result is returned, but it will cause QuickTime to report that it cannot open files that aren’t identified by a recognized file type or file name suffix.

kQTDontLookForMovieImporterIfGraphicsImporterFound

Tells QuickTime to short-circuit its search as soon as it finds one way to open the file. Pass this flag if you want to know whether a file can be opened with a graphics importer or as a movie, but you don’t care which.

kQTAllowOpeningStillImagesAsMovies

Tells QuickTime to consider opening still images as movies. If this flag is set, if a file can be opened using a graphics importer QuickTime will automatically say it can be opened as a movie.

CanQuickTimeOpenFile

kQTAllowImportersThatWouldCreateNewFile

Tells QuickTime to include importers which would create new files. If this flag is clear, QuickTime only includes importers which can import in place without needing to create new files.

kQTAllowAggressiveImporters

Tells QuickTime to include movie importers for file types like PICT and TEXT that aren't traditionally thought of as movies. If this flag is clear, QuickTime excludes these movie importers.

Discussion

This function determines whether QuickTime can open a given area of data. You should pass `NIL` in parameters that do not interest you, since that will allow QuickTime to perform a faster determination.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `Movies.h`

See Also

See `CanQuickTimeOpenFile` (I–72).

CanQuickTimeOpenFile

Determines whether a file can be opened using a graphics importer or opened in place as a movie.

```
OSErr CanQuickTimeOpenFile (
    FSSpecPtr    fileSpec,
    OSType       fileType,
    OSType       fileNameExtension,
    Boolean      *outCanOpenWithGraphicsImporter,
    Boolean      *outCanOpenAsMovie,
    Boolean      *outPreferGraphicsImporter,
    UInt32       inFlags );
```

fileSpec

Points to an `FSSpec` (IV–2262) structure that identifies a file. To ask about a particular file type or file name suffix in general, pass `NIL`.

fileType

Contains the file type if already known, or 0 if not known. If `fileSpec` is provided and `fileType` is 0, QuickTime will determine the file type. If you pass NIL in `fileSpec` and 0 in `fileNameExtension`, you must pass a file type here.

fileNameExtension

Contains the file name suffix if already known, or 0 if not known. The file name suffix should be encoded as an uppercase four character code with trailing spaces; for instance, the suffix ".png" should be encoded as 'PNG ', or 0x504e4720. If `fileSpec` is provided and `fileNameExtension` is 0, QuickTime will examine `fileSpec` to determine the file name suffix. If you pass NIL in `fileSpec` and 0 in `fileType`, you must pass a file name suffix here.

outCanOpenWithGraphicsImporter

Points to a Boolean that will be set to TRUE if the file can be opened using a graphics importer and FALSE otherwise. If you do not want this information, pass NIL.

outCanOpenAsMovie

Points to a Boolean that will be set to TRUE if the file can be opened as a movie and FALSE otherwise. If you do not want this information, pass NIL.

outPreferGraphicsImporter

Points to a boolean which will be set to true if the file can be opened using a graphics importer and opened as a movie, but, other factors being equal, QuickTime prefers a graphics importer. For example, QuickTime recommends using a graphics importer for single-frame GIF files and opening as a movie for multiple-frame GIF files. If you do not want this information, pass NIL. Passing a valid pointer disables the `kQTDontUseDataToFindImporter` and `kQTDontLookForMovieImporterIfGraphicsImporterFound` flags, if set.

inFlags

Flags (see below) that modify search behavior. Pass 0 for default behavior.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

inFlags Constants**kQTDontUseDataToFindImporter**

Tells QuickTime not to use the data in the file to help in the search. This will speed up the search, especially in cases where a negative result is returned, but it will cause QuickTime to report that it cannot open files that aren't identified by a recognized file type or file name suffix.

CaptureComponent

`kQTDontLookForMovieImporterIfGraphicsImporterFound`

Tells QuickTime to short-circuit its search as soon as it finds one way to open the file. Pass this flag if you want to know whether a file can be opened with a graphics importer or as a movie, but you don't care which.

`kQTAllowOpeningStillImagesAsMovies`

Tells QuickTime to consider opening still images as movies. If this flag is set, if a file can be opened using a graphics importer QuickTime will automatically say it can be opened as a movie.

`kQTAllowImportersThatWouldCreateNewFile`

Tells QuickTime to include importers which would create new files. If this flag is clear, QuickTime only includes importers which can import in place without needing to create new files.

`kQTAllowAggressiveImporters`

Tells QuickTime to include movie importers for file types like PICT and TEXT that aren't traditionally thought of as movies. If this flag is clear, QuickTime excludes these movie importers.

Discussion

This function determines whether QuickTime can open a given file or, in general, files of a given type. You should pass `NIL` in parameters that do not interest you, since that will allow QuickTime to perform a faster determination.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `Movies.h`

See Also

See `CanQuickTimeOpenDataRef` (I–70).

CaptureComponent

Allows your component to capture another component.

```
Component CaptureComponent (  
    Component    capturedComponent,  
    Component    capturingComponent );
```

`capturedComponent`

The identifier of the component to be captured. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`capturingComponent`

The identifier of your component. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result A new identifier that your software can use to refer to the captured component, or `NIL` if the component identified by `capturedComponent` is already captured.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

CDSequenceBusy

Checks the status of an asynchronous compression or decompression operation.

```
OSErr CDSequenceBusy (
    ImageSequence    seqID );
```

`seqID`

Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (I–238) or `CompressSequenceBegin` (I–119).

function result If there is no asynchronous operation in progress, `CDSequenceBusy` returns a 0 result code. If there is an asynchronous operation in progress, the result code is 1. Negative result codes indicate an error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V–2792)

Related Java Methods

`quicktime.std.image.CDSequence.busy()`

See Also

See `DecompressSequenceBegin` (I–238) and `CompressSequenceBegin` (I–119).

CDSequenceChangedSourceData

CDSequenceChangedSourceData

Notifies the compressor that the image source data has changed.

```
OSErr CDSequenceChangedSourceData (  
    ImageSequenceDataSource    sourceID );
```

sourceID

Contains the source identifier of the data source.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Use this function to indicate that the image has changed but the data pointer to that image has not changed. For example, if the data pointer points to the base address of a `PixelFormat` (IV–2332) structure, the image in the `PixelFormat` can change, but the data pointer remains constant.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V–2792)

Related Java Methods

`quicktime.std.image.ImageSequenceDataSource.changedSourceData()`

CDSequenceDisposeDataSource

Disposes of a data source.

```
OSErr CDSequenceDisposeDataSource (  
    ImageSequenceDataSource    sourceID );
```

sourceID

The data source identifier that was returned by the `CDSequenceNewDataSource` (I–82) function.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Use this function to dispose of a data source created by the `CDSequenceNewDataSource` (I–82) function. All data sources are automatically disposed when the sequence they are associated with is disposed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V–2792)

Related Java Methods

`quicktime.std.image.ImageSequenceDataSource.dispose()`

CDSequenceDisposeMemory

Disposes of memory allocated by the codec.

```
OSErr CDSequenceDisposeMemory (
    ImageSequence  seqID,
    Ptr            data );
```

seqID

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (I–238) function.

data

Points to the previously allocated memory block.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You call this function to release memory allocated by the `CDSequenceNewMemory` (I–84) function.

Special Considerations

Do not call `CDSequenceDisposeMemory` at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V–2792)

CDSequenceEnd

Indicates the end of processing for an image sequence.

CDSequenceEquivalentImageDescription

```
OSErr CDSequenceEnd (
    ImageSequence    seqID );
```

seqID

Contains the unique sequence identifier that was returned by
DecompressSequenceBegin (I–238) or CompressSequenceBegin (I–119).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Working With Sequences” (V–2792)

See Also

See DecompressSequenceBegin (I–238) and CompressSequenceBegin (I–119).

CDSequenceEquivalentImageDescription

Reports whether two image descriptions are the same.

```
OSErr CDSequenceEquivalentImageDescription (
    ImageSequence    seqID,
    ImageDescriptionHandle newDesc,
    Boolean          *equivalent );
```

seqID

Contains the unique sequence identifier that was returned by the
DecompressSequenceBegin (I–238) function.

newDesc

A handle to the ImageDescription (IV–2285) structure structure that describes
the compressed image.

equivalent

A pointer to a Boolean value. If the ImageDescriptionHandle provided in the
newDesc parameter is equivalent to the ImageDescription (IV–2285) structure
currently in use by the image sequence, this value is set to TRUE. If the
ImageDescriptionHandle is not equivalent, and therefore a new image sequence
must be created to display an image using the new image description, this
value is set to FALSE.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function allows an application to ask whether two image descriptions are the same. If they are, the decompressor does not have to create a new image decompression sequence to display those images.

Special Considerations

The Image Compression Manager can only implement part of this function by itself. There are some fields in the `ImageDescription` (IV–2285) structure that it knows are irrelevant to the decompressor. If the Image Compression Manager determines that there are differences in fields that may be significant to the codec, it calls the function `ImageCodecIsImageDescriptionEquivalent` to ask the codec.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V–2792)

Related Java Methods

`quicktime.std.image.CDSequence.equivalentImageDescription()`

See Also

See `ImageCodecIsImageDescriptionEquivalent` (II–725).

CDSequenceEquivalentImageDescriptionS

Undocumented

```
OSErr CDSequenceEquivalentImageDescriptionS (
    ImageSequence          seqID,
    ImageDescriptionHandle newDesc,
    Boolean                *equivalent,
    Boolean                *canSwitch );
```

`seqID`

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (I–238) function.

`newDesc`

A handle to the `ImageDescription` (IV–2285) structure structure that describes the compressed image.

`equivalent`

A pointer to a Boolean value. If the `ImageDescriptionHandle` provided in the `newDesc` parameter is equivalent to the `ImageDescription` (IV–2285) structure

CDSequenceFlush

currently in use by the image sequence, this value is set to `TRUE`. If the `ImageDescriptionHandle` is not equivalent, and therefore a new image sequence must be created to display an image using the new image description, this value is set to `FALSE`.

`canSwitch`

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCompression.h`

CDSequenceFlush

Stops a decompression sequence, aborting processing of any queued frames.

```
OSErr CDSequenceFlush (  
    ImageSequence    seqID );
```

`seqID`

Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (I–238).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is used to tell a decompressor component to stop processing of any queued scheduled asynchronous decompression. This is useful when several frames have been queued for decompression in the future and the application needs to suspend playback of the sequence.

For any outstanding frames, your application’s completion routine, passed to `DecompressSequenceFrameWhen` (I–245), will be called with an error result of `–1`, indicating that the frame was cancelled. If any frames are currently being decompressed and cannot be cancelled, `CDSequenceFlush` waits until the frame has finished decompressing before returning.

Version Notes

Introduced in QuickTime 2.0.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V–2792)

Related Java Methods

`quicktime.std.image.DSequence.flush()`

See Also

See `DecompressSequenceBegin` (I–238).

CDSequenceGetDataSource

Gets a data source for a decompression sequence.

```
OSErr CDSequenceGetDataSource (
    ImageSequence      seqID,
    ImageSequenceDataSource *sourceID,
    OSType              sourceType,
    long                 sourceInputNumber );
```

seqID

The image sequence that this source is associated with.

sourceID

A pointer to the source reference identifying this source.

sourceType

A four-character code describing how the input will be used. This value is passed by `CDSequenceNewDataSource` (I–82) when the source is created.

sourceInputNumber

A value passed by `CDSequenceNewDataSource` (I–82) when the source is created.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Related Java Methods

`quicktime.std.image.ImageSequenceDataSource.ImageSequenceDataSource()`

CDSequenceInvalidate

CDSequenceInvalidate

Notifies the Image Compression Manager that the destination port for the given image decompression sequence has been invalidated.

```
OSErr CDSequenceInvalidate (
    ImageSequence    seqID,
    RgnHandle        invalRgn );
```

seqID

Contains the unique sequence identifier that was returned by DecompressSequenceBegin (I-238).

invalRgn

A handle to the region specifying the invalid portion of the image.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Discussion

You call this function to force the Image Compression Manager to redraw the screen bits on the next decompression operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Working With Sequences” (V-2792)

Related Java Methods

quicktime.std.image.DSequence.invalidate()

See Also

See DecompressSequenceBegin (I-238).

CDSequenceNewDataSource

Creates a new data source.

```
OSErr CDSequenceNewDataSource (
    ImageSequence    seqID,
    ImageSequenceDataSource *sourceID,
    OSType           sourceType,
    long             sourceInputNumber,
    Handle            dataDescription,
    ICMConvertDataFormatUPP transferProc,
    void              *refCon );
```


seqID

The unique sequence identifier that was returned by the DecompressSequenceBegin (I-238) function.

sourceID

Returns the new data source identifier.

sourceType

A four-character code describing how the input will be used. This code is usually derived from the information returned by the codec. For example, if a mask plane was passed, this field might contain 'mask'.

sourceInputNumber

More than one instance of a given source type may exist. The first occurrence should have a source input number of 1, the second a source input number of 2, and so on.

dataDescription

A handle to a data structure describing the input data. For compressed image data, this is an ImageDescriptionHandle.

transferProc

A routine that allows the application to transform the type of the input data to the kind of data preferred by the codec. The client of the codec passes the source data in the form most convenient for it. If the codec needs the data in another form, it can negotiate with the client or directly with the Image Compression Manager to obtain the required data format.

refCon

A reference constant to be passed to the transfer procedure. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Discussion

This function returns a sourceID parameter which must be passed to all other functions that reference the source. All data sources are automatically disposed when the sequence they are associated with is disposed.

```
// CDSequenceNewDataSource coding example
// See “Discovering QuickTime,” page 309
{
    ImageSequenceDataSource    lSrc1 = 0;

    // Store a description of the first GWorld in hImageDesc1
    nErr = MakeImageDescriptionForPixMap(GetGWorldPixMap(gWorld1),
```

CDSequenceNewMemory

```
        &hImageDesc1);

    // Create a source from the GWorld description.
    nErr = CDSequenceNewDataSource(gEffectSequenceID,
                                   &lSrc1,
                                   'srcA',
                                   1,
                                   (Handle)hImageDesc1,
                                   NIL,
                                   0);

    // Set the data for source srcA to be the pixMap of gWorld1
    CDSequenceSetSourceData(lSrc1,
                            GetPixBaseAddr(GetGWorldPixMap(gWorld1)),
                            (**hImageDesc1).dataSize);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V-2792)

Related Java Methods

`quicktime.std.image.ImageSequenceDataSource.ImageSequenceDataSource()`

See Also

See `DecompressSequenceBegin` (I-238).

CDSequenceNewMemory

Requests codec-allocated memory.

```
OSErr CDSequenceNewMemory (
    ImageSequence      seqID,
    Ptr                *data,
    Size                dataSize,
    long                dataUse,
    ICMemoryDisposedUPP memoryGoneProc,
    void                *refCon );
```

seqID

Contains the unique sequence identifier that was returned by the DecompressSequenceBegin (I–238) function.

data

Returns a pointer to the allocated memory.

dataSize

The requested size of the data buffer.

dataUse

A code (see below) that indicates how the memory is to be used. For example, the memory may be used to store compressed image or mask plane data, or used as an offscreen image buffer. If there is no benefit to storing a particular kind of data in codec memory, the codec should deny the request for the memory allocation.

memoryGoneProc

A pointer to a callback function that will be called before disposing of the memory allocated by a codec, as described in ICMMemoryDisposedProc (III–2092).

refCon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

dataUse Constants

0x0001

Memory will be used for holding compressed image data.

0x0002

Memory will be used for an offscreen image buffer.

Discussion

Because many hardware decompression boards contain dedicated on-board memory, significant performance gains can be realized if this memory is used to store data before it is decompressed. When memory is allocated, a callback function must be provided, as described in ICMMemoryDisposedProc. The decompressor can dispose of all memory it has allocated at any time, but it calls the callback routine before disposing of the memory. A callback procedure is required because memory on the hardware decompression board may be limited. If the decompressor cannot deallocate memory as required, it is possible that an idle decompressor instance may be holding a large amount of memory, denying those **resources** to the currently active decompressor instance. When the callback procedure is called, the memory is still available. This allows any pending reads into the block to be canceled before the block is disposed. The decompressor disposing the memory must ensure that it

CDSequenceSetSourceData

is not disposing a block that it is currently using (that is, a block that contains the currently decompressing frame). To dispose of the memory, use `CDSequenceDisposeMemory` (I-77).

Special Considerations

Decompressor memory must never be disposed at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V-2792)

See Also

See `CDSequenceDisposeMemory` (I-77). For a specification of your callback function, see `ICMemoryDisposedProc` (III-2092).

CDSequenceSetSourceData

Sets data in a new frame to a specific data source.

```
OSErr CDSequenceSetSourceData (
    ImageSequenceDataSource  sourceID,
    void                    *data,
    long                    dataSize );
```

sourceID

Contains the source identifier of the data source.

data

Points to the data. This pointer must contain a **32-bit clean** address.

dataSize

The size of the data buffer.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This function is called to set data in a new frame to a specific source. For example, as a new frame of compressed data arrives at a source, `CDSequenceSetSourceData` will be called.

```
// CDSequenceSetSourceData coding example
// See “Discovering QuickTime,” page 309
{
```

```

ImageSequenceDataSource    lSrc1 = 0;

// Store a description of the first GWorld in hImageDesc1
nErr = MakeImageDescriptionForPixMap(GetGWorldPixMap(gWorld1),
                                     &hImageDesc1);

// Create a source from the GWorld description.
nErr = CDSequenceNewDataSource(gEffectSequenceID,
                              &lSrc1,
                              'srcA',
                              1,
                              (Handle)hImageDesc1,
                              NIL,
                              0);

// Set the data for source srcA to be the pixMap of gWorld1
CDSequenceSetSourceData(lSrc1,
                       GetPixBaseAddr(GetGWorldPixMap(gWorld1)),
                       (**hImageDesc1).dataSize);
}

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Working With Sequences” (V-2792)

Related Java Methods

quicktime.std.image.ImageSequenceDataSource.setSourceData()

CDSequenceSetSourceDataQueue

Sets a data queue as the source for a decompression sequence.

```

OSErr CDSequenceSetSourceDataQueue (
    ImageSequenceDataSource    sourceID,
    QHdrPtr                   dataQueue );

```

sourceID

Contains the source identifier of the data source.

CDSequenceSetTimeBase

`dataQueue`

A pointer to a QHdr (IV–2345) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

CDSequenceSetTimeBase

Sets a time base for a decompression sequence.

```
OSErr CDSequenceSetTimeBase (  
    ImageSequence    seqID,  
    void             *base );
```

`seqID`

A unique sequence identifier that was returned by `CompressSequenceBegin` (I–119).

`base`

A pointer to the time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II–1119).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

When you run a visual effect outside a movie, you must designate a time base that will be used when the effect is run. The following code illustrates this use of `CDSequenceSetTimeBase`:

```
// CDSequenceSetTimeBase coding example  
// See “Discovering QuickTime,” page 310  
timeBase = NewTimeBase();  
SetTimeBaseRate(timeBase, 0);  
CDSequenceSetTimeBase(gEffectSequenceID, timeBase);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Related Java Methods

`quicktime.std.image.CDSequence.setTimeBase()`

CheckQuickTimeRegistration

Deprecated.

```
void CheckQuickTimeRegistration (
    void    *registrationKey,
    long    flags );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime.

Programming Info

C interface file: `Movies.h`

ClearMovieChanged

Sets the movie changed flag to indicate that the movie has not been changed.

```
void ClearMovieChanged (
    Movie    theMovie );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Saving Movies” (V–2715)

Related Java Methods

`quicktime.std.movies.Movie.clearChanged()`

ClearMovieSelection

ClearMovieSelection

Removes the segment of the movie that is defined by the current selection.

```
void ClearMovieSelection (
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Movie Editing Functions” (V–2746)

Related Java Methods

`quicktime.std.movies.Movie.clearSelection()`

ClearMoviesStickyError

Clears the **sticky error** value.

```
void ClearMoviesStickyError ( void );
```

Discussion

The Movie Toolbox provides two error values to your application: the current error and the **sticky error**. The current error is the result code from the last Movie Toolbox function; it is updated each time your application calls a Movie Toolbox function. The Movie Toolbox saves the same result code in the sticky error value. Your application clears the sticky error value by calling `ClearMoviesStickyError`. The Movie Toolbox then places the first nonzero result code from any toolbox function used by your application into the sticky error value. The Movie Toolbox does not update the sticky error value until your application clears it again.

Special Considerations

Many Movie Toolbox functions don’t return an error as a function result; you must use `GetMoviesError` to obtain the result code. Even if a function explicitly returns an

error as a function result, that result is also available using `GetMoviesError`. The Movie Toolbox does not place a result code into the **sticky error** value until the field has been cleared. Your application is responsible for clearing the sticky error value to ensure that it does not contain a stale result code.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Error Functions” (V-2712)

See Also

Your application calls `GetMoviesError` (I-492) to obtain the current error value after calling any Movie Toolbox function. Your application calls `GetMoviesStickyError` (I-494) to obtain the result code stored in the **sticky error** value.

ClockCallMeWhen

In a clock component, schedules a **callback event** for invocation.

```
ComponentResult ClockCallMeWhen (
    ComponentInstance  aClock,
    QTCallBack         cb,
    long               param1,
    long               param2,
    long               param3 );
```

`aClock`

Specifies the clock for the operation. Applications obtain this identifier from `OpenComponent` (II-1130).

`cb`

Specifies the **callback event** for the operation. The Movie Toolbox obtains this value from your component’s `ClockNewCallBack` (I-96) function.

`param1`

Contains data supplied to the Movie Toolbox in the `param1` parameter to the `CallMeWhen` (I-67) function. Your component interprets this parameter based on the value of the `callbackType` parameter to the `ClockNewCallBack` (I-96) function. If `callbackType` is set to `callbackAtTime`, the `param1` parameter contains flags (see below) indicating when to invoke your callback function for this **callback event**. If the `callbackType` parameter is set to `callbackAtRate`, `param1`

ClockCallMeWhen

contains flags (see below) indicating when to invoke your callback function for this event.

param2

Contains data supplied to the Movie Toolbox in the param2 parameter to the CallMeWhen (I-67) function. Your component interprets this parameter based on the value of the callbackType parameter to the ClockNewCallback (I-96) function. If callbackType is set to callbackAtTime, the param2 parameter contains the time value at which your callback function is to be invoked for this event. The param1 parameter contains flags affecting when the Movie Toolbox calls your function. If callbackType is set to callbackAtRate, the param2 parameter contains the rate value at which your callback function is to be invoked for this event.

param3

Contains data supplied to the Movie Toolbox in the param3 parameter to the CallMeWhen (I-67) function. If cbType is set to callbackAtTime, param3 contains the time scale in which to interpret the time value that is stored in param2.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

callbackAtTime Constants

triggerTimeFwd

Indicates that your callback function should be called at the time specified by param2 only when time is moving forward (positive rate). The value of this flag is 0x0001.

triggerTimeBwd

Indicates that your callback function should be called at the time specified by param2 only when time is moving backward (negative rate). The value of this flag is 0x0002.

triggerTimeEither

Indicates that your callback function should be called at the time specified by param2 without regard to direction, but the rate must be nonzero. The value of this flag is 0x0003.

callbackAtRate Constants

triggerRateChange

Indicates that your callback function should be called whenever the rate changes. The value of this flag is 0x0000.

triggerRateLT

Indicates that your callback function should be called when the rate changes to a value less than that specified by param2. The value of this flag is 0x0004.

`triggerRateGT`

Indicates that your callback function should be called when the rate changes to a value greater than that specified by `param2`. The value of this flag is 0x0008.

`triggerRateEqual`

Indicates that your callback function should be called when the rate changes to a value equal to that specified by `param2`. The value of this flag is 0x0010.

`triggerRateLTE`

Indicates that your callback function should be called when the rate changes to a value that is less than or equal to that specified by `param2`. The value of this flag is 0x0014.

`triggerRateGTE`

Indicates that your callback function should be called when the rate changes to a value that is greater than or equal to that specified by `param2`. The value of this flag is 0x0018.

`triggerRateNotEqual`

Indicates that your callback function should be called when the rate changes to a value that is not equal to that specified by `param2`. The value of this flag is 0x001C.

Discussion

If your clock component successfully schedules the **callback event**, you should call the `AddCallbackToTimeBase` (I-21) function to add it to the list of callback events for the corresponding time base. If your component cannot schedule the callback event, it should return an appropriate error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Using Callback Functions” (V-2867)

See Also

See `ClockNewCallback` (I-96) and `CallMeWhen` (I-67).

ClockCancelCallback

In a clock component, removes the specified **callback event** from the list of scheduled callback events for a time base.

```
ComponentResult ClockCancelCallback (
    ComponentInstance    aClock,
```

ClockDisposeCallback

```
QTCallback cb );
```

aClock

Specifies the clock for the operation. Your application obtains this identifier from the Component Manager's `OpenComponent` (II–1130) function.

cb

Specifies the **callback event** for the operation. The Movie Toolbox obtains this value from your component's `ClockNewCallback` (I–96) function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If your clock component successfully cancels the **callback event**, you should call the `RemoveCallbackFromTimeBase` (III–1456) function so that the Movie Toolbox can remove the callback event from its list of scheduled events.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Using Callback Functions” (V–2867)

ClockDisposeCallback

In a clock component, disposes of the memory associated with the specified **callback event**.

```
ComponentResult ClockDisposeCallback (  
    ComponentInstance aClock,  
    QTCallback cb );
```

aClock

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager's `OpenComponent` (II–1130) function.

cb

Specifies the **callback event** for the operation. The Movie Toolbox obtains this value from your component's `ClockNewCallback` (I–96) function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You should not call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Using Callback Functions” (V–2867)

ClockGetRate

Fetches the rate of a specified clock.

```
ComponentResult ClockGetRate (
    ComponentInstance    aClock,
    Fixed                *rate );
```

aClock

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager’s `OpenComponent` (II–1130) function.

rate

Pointer to memory where the clock rate is returned.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

ClockGetTime

Obtains the current time according to a specified clock.

```
ComponentResult ClockGetTime (
    ComponentInstance    aClock,
    TimeRecord           *out );
```

aClock

Specifies the clock for the operation. You obtain this identifier from the Component Manager’s `OpenComponent` (II–1130) function. See *Inside Macintosh: More Macintosh Toolbox* (listed in the **bibliography**) for details.

ClockNewCallBack

out

A pointer to a `TimeRecord` (IV–2486) structure. The clock component updates this structure with the current time information. Specifically, the clock component sets the `value` field and the `scale` field in the **time structure**. Your clock component should always return values in its native time scale. This time scale does not change during the life of the component connection.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Getting the Current Time” (V–2867)

Related Java Methods

`quicktime.std.clocks.Clock.getTime()`

ClockNewCallBack

In a clock component, allocates memory for a new **callback event**.

```
QTCallBack ClockNewCallBack (
    ComponentInstance    aClock,
    TimeBase              tb,
    short                 callBackType );
```

`aClock`

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager’s `OpenComponent` (II–1130) function.

`tb`

Specifies the **callback event**’s time base. Typically, your component does not need to save this specification. You can use the Movie Toolbox’s `GetCallBackTimeBase` (I–384) function to determine the callback event’s time base when it is invoked. For more information about time bases, see *Inside Macintosh: QuickTime* (listed in the **bibliography**).

`callBackType`

Contains a constant (see below) that specifies when the **callback event** is to be invoked. The value of this parameter governs how your component interprets the data supplied in the `param1`, `param2`, and `param3` parameters to `ClockCallMeWhen` (I–91).

function result A pointer to a `CallBackRecord` (IV–2165) structure. Your software can pass this structure to other functions, such as `ClockRateChanged` (I–97).

callbackType Constants

`callbackAtTime`

Indicates that the event is to be invoked at a specified time.

`callbackAtRate`

Indicates that the event is to be invoked when the rate for the time base reaches a specified value.

`callbackAtTimeJump`

Indicates that the event is to be invoked when the time base’s time value changes by an amount that differs from its rate.

`callbackAtInterrupt`

Defines the bit of the returned value which indicates that the event can be invoked at interrupt time.

Discussion

Your component allocates the memory required to support the **callback event**. The memory must be in a locked block and must begin with a `QTCallBackHeader` (IV–2349) structure initialized to 0. Your component can allocate an arbitrarily large piece of memory for the callback event.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Using Callback Functions” (V–2867)

ClockRateChanged

In a clock component, is called whenever the callback’s time base rate changes.

```
ComponentResult ClockRateChanged (
    ComponentInstance  aClock,
    QTCallBack         cb );
```

`aClock`

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager’s `OpenComponent` (II–1130) function.

ClockSetTimeBase

`cb`

Specifies the callback for the operation. The Movie Toolbox obtains this value from your component's `ClockNewCallback` (I-96) function.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

The Movie Toolbox calls this function once for each qualified callback function associated with the time base. Note that the Movie Toolbox calls this function only for **callback events** that are currently scheduled.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing the Time” (V-2868)

See Also

See the `CallbackRecord` (IV-2165) structure.

ClockSetTimeBase

In a clock component, is called when an application creates a time base that uses the clock component.

```
ComponentResult ClockSetTimeBase (  
    ComponentInstance    aClock,  
    TimeBase             tb );
```

`aClock`

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager's `OpenComponent` (II-1130) function.

`tb`

Specifies the time base that is associated with the clock.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing the Time” (V-2868)

ClockStartStopChanged

In a clock component, is called whenever the start or stop time of the callback's time base changes.

```
ComponentResult ClockStartStopChanged (
    ComponentInstance    aClock,
    QTCallback           cb,
    Boolean              startChanged,
    Boolean              stopChanged );
```

aClock

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager's `OpenComponent` (II-1130) function.

cb

Specifies the callback for the operation. The Movie Toolbox obtains this value from your component's `ClockNewCallback` (I-96) function.

startChanged

Indicates that the start time of the time base associated with the clock component instance has changed.

stopChanged

Indicates that the stop time of the time base associated with the clock component instance has changed.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

The Movie Toolbox calls this function once for each qualified callback function associated with the time base. Note that the Movie Toolbox calls this function only for **callback events** that are currently scheduled.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: "Managing the Time" (V-2868)

See Also

See the `CallbackRecord` (IV-2165) structure.

ClockTimeChanged

ClockTimeChanged

In a clock component, is called whenever the callback's time base time value is set.

```
ComponentResult ClockTimeChanged (  
    ComponentInstance    aClock,  
    QTCallback           cb );
```

aClock

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager's `OpenComponent` (II–1130) function.

cb

Specifies the callback for the operation. The Movie Toolbox obtains this value from your component's `ClockNewCallback` (I–96) function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing the Time” (V–2868)

See Also

See the `CallbackRecord` (IV–2165) structure.

CloseComponent

Terminates your application's access to the services provided by a component.

```
OSErr CloseComponent (  
    ComponentInstance    aComponentInstance );
```

aComponentInstance

A component instance. Your application obtains the component instance from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The following snippet shows how `CloseComponent` is called when a component is no longer needed.

```
// CloseComponent coding example
```

```
// See "Discovering QuickTime," page 279
void writeGWorldToImageFile(GWorldPtr gw, const FSSpec *fileSpec)
{
    GraphicsExportComponent ge = 0;

    OpenADefaultComponent(GraphicsExporterComponentType,
                           kQTFileTypePNG, &ge);
    GraphicsExportSetInputGWorld(ge, gw);
    GraphicsExportSetOutputFile(ge, fileSpec);
    GraphicsExportDoExport(ge, NIL);
    CloseComponent(ge);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: "Component Functions Used By Applications" (V-2781)

CloseComponentResFile

Closes the **resource** file that your component opened previously with the `OpenComponentResFile` function.

```
OSErr CloseComponentResFile (
    short    refnum );
```

refnum

A reference number that identifies the **resource** file. Your component obtains this value from `OpenComponentResFile` (II-1130).

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: "Functions Used By Components" (V-2782)

CloseMixerSoundComponent

CloseMixerSoundComponent

Closes the Apple Mixer component.

```
OSErr CloseMixerSoundComponent (  
    ComponentInstance  ci );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is usually called by a sound output device.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

CloseMovieFile

Closes an open movie file.

```
OSErr CloseMovieFile (  
    short    resRefNum );
```

resRefNum

The movie file to close. Your application obtains this reference number from `OpenMovieFile` (II–1133).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The following code shows a typical use of `CloseMovieFile`.

```
// CloseMovieFile coding example  
// See “Discovering QuickTime,” page 50  
void OpenMovie (HWND hwnd, char *szFileName)  
{  
    short    nFileRefNum = 0;
```

```

FSSpec fss;
// Convert path to FSSpec
NativePathNameToFSSpec(szFileName, &fss, 0);
// Set graphics port
SetGWorld((CGrafPtr)GetNativeWindowPort(hwnd), NIL);

OpenMovieFile(&fss, &nFileRefNum, fsRdPerm); // Open movie file
NewMovieFromFile(&movie, nFileRefNum, NIL, // Get movie from file
                NIL, newMovieActive, NIL);
CloseMovieFile(nFileRefNum); // Close movie file
}

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Functions” (V–2713)

Related Java Methods

`quicktime.io.OpenedFile.close()`

CodecManagerVersion

Determines the version of the installed Image Compression Manager.

```

OSErr CodecManagerVersion (
    long    *version );

```

version

A pointer to a long integer that is to receive the version information. The Image Compression Manager returns its version number into this location. The version number is a long integer value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns the version information as a long integer value. Use this function to retrieve the version number associated with the Image Compression Manager that is installed on a particular computer.

Special Considerations

The Image Compression Manager provides a number of functions that allow your application to obtain information about the facilities available for image

Comp3to1

compression or about compressed images. Your application may use some of these functions to select a specific compressor or decompressor for a given operation or to determine how much memory to allocate to receive a decompressed image. In addition, your application may use some of these functions to determine the capabilities of the components that are available on the user's computer system. You can then condition the options your program makes available to the user based on the user's system configuration.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: "Getting Information About Compressor Components" (V-2788)

Related Java Methods

`quicktime.std.image.CodecInfo.codecVersion()`

Comp3to1

Compresses sound samples to one-third their original size. Use `SoundConverterConvertBuffer` (III-1862) instead, if possible.

```
void Comp3to1 (
    const void      *inBuffer,
    void            *outBuffer,
    unsigned long    cnt,
    StateBlockPtr    inState,
    StateBlockPtr    outState,
    unsigned long    numChannels,
    unsigned long    whichChannel );
```

`inBuffer`

A pointer to a buffer of samples to be compressed.

`outBuffer`

A pointer to a buffer where the samples are to be written.

`cnt`

The number of samples to compress.

`inState`

A pointer to a 128-byte buffer from which the input state of the algorithm is read, or `NIL`. This buffer should be filled with zeros to initialize the algorithm.

`outState`

A pointer to a 128-byte buffer to which the output state of the algorithm is written, or `NIL`. This buffer may be the same as that specified by the `inState` parameter.

`numChannels`

The number of channels in the buffer pointed to by the `inBuffer` parameter.

`whichChannel`

The channel to compress when `numChannels` is greater than 1. The value of this parameter must be in the range 1 to `numChannels`.

Discussion

This function compresses `cnt` samples of sound stored in the buffer specified by `inBuffer` and places the result in the buffer specified by `outBuffer`, which must be at least `cnt/3` bytes in size. The original samples can be monophonic or include multiple channels of sound, but they must be in 8-bit offset binary format. Also, if `numChannels` is greater than 1, then the noncompressed sound must be stored in interleaved format on a sample basis. If you compress polyphonic sound, you retain only one channel of sound, which you specify in the `whichChannel` parameter. Thus, if you use the `Comp3to1` procedure to compress three-channel sound, you will have effectively compressed the sound to one-ninth its original size in bytes. To retain multiple channels of sound after compression, you must call the `Comp3to1` procedure for each channel to be compressed and then interleave the compressed sound data on a packet basis. The `Comp3to1` procedure compresses every 48 bytes of sound data to exactly 16 bytes of compressed sound data and compresses remaining bytes to no more than one-third the original size. You can use the `inState` and `outState` parameters to allow the **MACE** compression routines to preserve information about algorithms across calls. Alternatively, you may pass `NIL` state buffers and let the Sound Manager allocate the buffers internally.

Special Considerations

Because the `Comp3to1` procedure might allocate and dispose of memory, you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 4. Macintosh Note: Not available in **CarbonLib**, but available when InterfaceLib 7.1 or later is installed.

Programming Info

C interface file: `Sound.h`

Comp6to1

Comp6to1

Compresses sound samples to one-sixth their original size. Use `SoundConverterConvertBuffer` (III–1862) instead, if possible.

```
void Comp6to1 (
    const void      *inBuffer,
    void            *outBuffer,
    unsigned long    cnt,
    StateBlockPtr    inState,
    StateBlockPtr    outState,
    unsigned long    numChannels,
    unsigned long    whichChannel );
```

`inBuffer`

A pointer to a buffer of samples to be compressed.

`outBuffer`

A pointer to a buffer where the samples are to be written.

`cnt`

The number of samples to compress.

`inState`

A pointer to a 128-byte buffer from which the input state of the algorithm is read, or NIL. This buffer should be filled with zeros to initialize the algorithm.

`outState`

A pointer to a 128-byte buffer to which the output state of the algorithm is written, or NIL. This buffer may be the same as that specified by the `inState` parameter.

`numChannels`

The number of channels in the buffer pointed to by the `inBuffer` parameter.

`whichChannel`

The channel to compress when `numChannels` is greater than 1. The value of this parameter must be in the range 1 to `numChannels`.

Discussion

This function compresses `cnt` samples of sound stored in the buffer specified by `inBuffer` and places the result in the buffer specified by `outBuffer`, which must be at least `cnt/6` bytes in size. `Comp6to1` works much as `Comp3to1` (I–104), but compresses every 48 bytes of sound data to exactly 8 bytes of compressed sound data and compresses remaining bytes to no more than one-sixth the original size.

Special Considerations

Because the `Comp6to1` procedure might allocate and dispose of memory, you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 4. Macintosh Note: Not available in **CarbonLib**, but available when InterfaceLib 7.1 or later is installed.

Programming Info

C interface file: `Sound.h`

See Also

See `Comp3to1` (I–104).

CompAdd

Undocumented

```
void CompAdd (
    wide  *src,
    wide  *dst );
```

`src`

Undocumented

`dst`

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

CompCompare

Undocumented

```
long CompCompare (
    const wide  *a,
    const wide  *minusb );
```

`a`

Undocumented

CompDiv

minusb

Undocumented

function result *Undocumented*

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

CompDiv

Undocumented

```
long CompDiv (  
    wide      *numerator,  
    long      denominator,  
    long      *remainder );
```

numerator

Undocumented

denominator

Undocumented

remainder

Undocumented

function result *Undocumented*

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

CompFixMul

Undocumented

```
void CompFixMul (  
    wide      *compSrc,  
    Fixed     fixSrc,  
    wide      *compDst );
```

compSrc
Undocumented
 fixSrc
Undocumented
 compDst
Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

CompMul

Undocumented

```
void CompMul (
    long    src1,
    long    src2,
    wide    *dst );
```

src1
Undocumented
 src2
Undocumented
 dst
Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

CompMulDiv

Undocumented

```
void CompMulDiv (
    wide    *co,
    long    mul,
```

CompMulDivTrunc

```
long divisor );  
  
co  
    Undocumented  
mul  
    Undocumented  
divisor  
    Undocumented
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

CompMulDivTrunc

```
Undocumented  
  
void CompMulDivTrunc (  
    wide *co,  
    long mul,  
    long divisor,  
    long *remainder );  
  
co  
    Undocumented  
mul  
    Undocumented  
divisor  
    Undocumented  
remainder  
    Undocumented
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

CompNeg

Undocumented

```
void CompNeg (
    wide     *dst );
```

dst

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

C

ComponentFunctionImplemented

Determines whether a component supports a specified request.

```
long ComponentFunctionImplemented (
    ComponentInstance  ci,
    short              ftnNumber );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131).

ftnNumber

A request code value. See *Inside Macintosh: QuickTime Components* (listed in the **bibliography**) for information about the request codes supported by the QuickTime components supplied by Apple. For other components, see their developer's documentation.

function result A long integer that can be interpreted as a Boolean value. If TRUE, the component supports the request; if FALSE, it does not.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Components.h

Programming summary: "Component Functions Used By Applications" (V-2781)

ComponentSetTarget

ComponentSetTarget

Calls a component's target request routine, which handles the `kComponentTargetSelect` request code.

```
long ComponentSetTarget (
    ComponentInstance  ci,
    ComponentInstance  target );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

target

The component instance of the component issuing the target request.

function result The value that the targeted component instance returned in response to the target request, or a constant (see below).

Function Result Constants

`badComponentSelector`

The targeted component does not support the target request.

Discussion

This function returns a function result of `badComponentSelector` if the targeted component does not support the target request. Otherwise, the `ComponentSetTarget` function returns as its function result the value that the targeted component instance returned in response to the target request.

Special Considerations

You should not target a component instance if the component does not support the target request. Before calling this function, you should issue a can do request to the component instance you want to target to verify that the component supports the target request. If the component supports it, use the `ComponentSetTarget` function to send a target request to the component instance you wish to target. After receiving a target request, the targeted component instance should call the component instance that targeted it whenever the targeted component instance would normally call one of its defined functions.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

CompressImage

Compresses a single-frame image that is currently stored as a pixel map structure.

```
OSErr CompressImage (
    PixMapHandle      src,
    const Rect        *srcRect,
    CodecQ            quality,
    CodecType         cType,
    ImageDescriptionHandle desc,
    Ptr               data );
```

src

A handle to the image to be compressed. The image must be stored in a pixel map structure.

srcRect

A pointer to a rectangle defining the portion of the image to compress.

quality

A constant (see below) that defines the desired compressed image quality.

cType

A compressor type; see “Codec Identifiers” (IV–2655).

desc

A handle that is to receive a formatted `ImageDescription` (IV–2285) structure. The Image Compression Manager resizes this handle for the returned image description structure. Your application should store this image description with the compressed image data.

data

Points to a location to receive the compressed image data. It is your program’s responsibility to make sure that this location can receive at least as much data as indicated by the `GetMaxCompressionSize` (I–420) function. The Image Compression Manager places the actual size of the compressed image into the `dataSize` field of the `ImageDescription` (IV–2285) structure referred to by the `desc` parameter. This pointer must contain a **32-bit clean** address.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

quality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

CompressImage

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

The following code sample illustrates the process of compressing and decompressing a pixel map.

```
// CompressImage coding example
// See "Discovering QuickTime," page 286
PicHandle GetQTCompressedPict (PixMapHandle hpmImage)
{
    long                lMaxCompressedSize = 0;
    Handle              hCompressedData = NIL;
    Ptr                 pCompressedData;
    ImageDescriptionHandle hImageDesc = NIL;
    OSErr               nErr;
    PicHandle           hpicPicture = NIL;
    Rect                rectImage = (**hpmImage).bounds;
    CodecType           dwCodecType = kJPEGCodecType;
    CodecComponent       codec = (CodecComponent)anyCodec;
    CodecQ              dwSpatialQuality = codecNormalQuality;
    short               nDepth = 0;          // let ICM choose depth

    nErr = GetMaxCompressionSize(hpmImage, &rectImage, nDepth,
                                dwSpatialQuality,
                                dwCodecType,
                                (CompressorComponent)codec,
                                &lMaxCompressedSize);

    if (nErr != noErr)
        return NIL;
```



```

hImageDesc = (ImageDescriptionHandle)NewHandle(4);
hCompressedData = NewHandle(1MaxCompressedSize);
if ((hCompressedData != NIL) && (hImageDesc != NIL)) {
    MoveHHi(hCompressedData);
    HLock(hCompressedData);
    pCompressedData = StripAddress(*hCompressedData);

    nErr = CompressImage(hpmImage,
                        &rectImage,
                        dwSpatialQuality,
                        dwCodecType,
                        hImageDesc,
                        pCompressedData);

    if (nErr == noErr) {
        ClipRect(&rectImage);
        hpicPicture = OpenPicture(&rectImage);
        nErr = DecompressImage(pCompressedData,
                              hImageDesc,
                              hpmImage,
                              &rectImage,
                              &rectImage,
                              srcCopy,
                              NIL);

        ClosePicture();
    }

    if (theErr || (GetHandleSize((Handle)hpicPicture) ==
                  sizeof(Picture))) {
        KillPicture(hpicPicture);
        hpicPicture = NIL;
    }
}

if (hImageDesc != NIL)
    DisposeHandle((Handle)hImageDesc);

if (hCompressedData != NIL)

```

CompressPicture

```
        DisposeHandle(hCompressedData);

    return hpicPicture;
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pixel Maps” (V–2789)

Related Java Methods

`quicktime.std.image.QTImage.compress()`

CompressPicture

Compresses a single-frame image stored as a picture structure and places the result in another picture.

```
OSErr CompressPicture (
    PicHandle    srcPicture,
    PicHandle    dstPicture,
    CodecQ       quality,
    CodecType    cType );
```

`srcPicture`

A handle to the source image, stored as a picture.

`dstPicture`

A handle to the destination for the compressed image. The compressor resizes this handle for the result data.

`quality`

A constant (see below) that defines the desired compressed image quality.

`cType`

You must set this parameter to a valid compressor identifier; see “Codec Identifiers” (IV–2655). If the value passed in is 0, or ‘raw’, and the source picture is compressed, the destination picture is created as an uncompressed picture and does not require QuickTime for its display.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

quality Constants`codecMinQuality`

The minimum valid value for a CodecQ field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a CodecQ field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

This function compresses only image data. Any other types of data in the picture, such as text, graphics primitives, and previously compressed images, are not modified in any way and are passed through to the destination picture. This function supports parameters governing image quality and compressor type. The compressor infers the other compression parameters from the image data in the source picture.

Special Considerations

If a picture with multiple pixel maps and other graphical objects is passed, the pixel maps will be compressed individually and the other graphic objects will not be affected. This function does not use the graphics port.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pictures and PICT Files” (V-2790)

Related Java Methods

`quicktime.qd.Pict.compress()`

See Also

If your picture does not fit in memory, use the `CompressPictureFile` (I-118) function.

CompressPictureFile

Compresses a single-frame image stored as a **picture file** and places the result in another picture file.

```
OSErr CompressPictureFile (  
    short      srcRefNum,  
    short      dstRefNum,  
    CodecQ     quality,  
    CodecType  cType );
```

srcRefNum

A file reference number for the source 'PICT' file.

dstRefNum

A file reference number for the destination 'PICT' file. Note that the compressor overwrites the contents of the file referred to by *dstRefNum*. You must open this file with write permission. The destination file can be the same as the source file specified by the *srcRefNum* parameter.

quality

A constant (see below) that defines the desired compressed image quality.

cType

A compressor type. You must set this parameter to a valid compressor type constant; see “Codec Identifiers” (IV–2655). If the value passed in is 0, or 'raw', and the source picture is compressed, the destination picture is created as an uncompressed picture and does not require QuickTime to be displayed.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

quality Constants

codecMinQuality

The minimum valid value for a *CodecQ* field.

codecLowQuality

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

codecNormalQuality

Image reproduction of normal quality.

codecHighQuality

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

codecMaxQuality

The maximum standard value for a *CodecQ* field.

codeLosslessQuality

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

This function supports parameters governing image quality and compressor type. The compressor infers the other compression parameters from the image data in the source **picture file**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Working With Pictures and PICT Files” (V-2790)



CompressSequenceBegin

Signals the beginning of the process of compressing a sequence of frames.

```
OSErr CompressSequenceBegin (
    ImageSequence          *seqID,
    PixMapHandle           src,
    PixMapHandle           prev,
    const Rect             *srcRect,
    const Rect             *prevRect,
    short                  colorDepth,
    CodecType              cType,
    CompressorComponent    codec,
    CodecQ                 spatialQuality,
    CodecQ                 temporalQuality,
    long                   keyFrameRate,
    CTabHandle             ctable,
    CodecFlags             flags,
    ImageDescriptionHandle desc );
```

seqID

A pointer to a field to receive the unique identifier for this sequence. You must supply this identifier to all subsequent Image Compression Manager functions that relate to this sequence.

src

A handle to a pixel map that will contain the image to be compressed. The image must be stored in a pixel map structure.

CompressSequenceBegin

prev

A handle to a pixel map that will contain a previous image. The compressor uses this buffer to store a previous image against which the current image (stored in the pixel map referred to by the `src` parameter) is compared when performing temporal compression. This pixel map must be created at the same depth and with the same color table as the source image. The compressor manages the contents of this pixel map based upon several considerations, such as the **key frame** rate and the degree of difference between compared images. If you want the compressor to allocate this pixel map or if you do not want to perform temporal compression (that is, you have set the value of the `temporalQuality` parameter to 0), set this parameter to `NIL`.

srcRect

A pointer to a rectangle defining the portion of the image to compress. The compressor applies this rectangle to the image stored in the buffer referred to by the `src` parameter.

prevRect

A pointer to a rectangle defining the portion of the previous image to use for temporal compression. The compressor uses this portion of the previous image as the basis of comparison with the current image. The compressor ignores this parameter if you have not provided a buffer for previous images. This rectangle must be the same size as the source rectangle, which is specified with the `srcRect` parameter.

colorDepth

The depth at which the sequence is likely to be viewed. Compressors may use this as an indication of the color or grayscale resolution of the compressed images. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the `GetCodecInfo` (I-385) function.

cType

You must set this parameter to a valid compressor type constant. See “Codec Identifiers” (IV-2655).

codec

Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below).

Specifying a component instance may be useful if you have previously set some

parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation.

spatialQuality

A pointer to a field containing a constant (see below) that defines the desired compressed image quality. You can change the value of this parameter for an active sequence by calling `SetCSequenceQuality` (III–1580).

temporalQuality

A pointer to a field containing a constant (see below) that defines the desired temporal quality. This parameter governs the level of compression you desire with respect to information between successive frames in the sequence. Set to 0 if you do not want temporal compression. You can change the value of this parameter for an active sequence by calling `SetCSequenceQuality` (III–1580).

keyFrameRate

Specifies the maximum number of frames allowed between **key frames**. The compressor determines the optimum placement for key frames based upon the amount of redundancy between adjacent images in the sequence. Consequently, the compressor may insert key frames more frequently than you have requested. However, the compressor never places fewer key frames than is indicated by the setting of the `keyFrameRate` parameter. The compressor ignores this parameter if you have not requested temporal compression (that is, you have set the `temporalQuality` parameter to 0). If you pass in 0 in this parameter, this indicates that there are no key frames in the sequence. If you pass in any other number, it specifies the number of non-key frames between key frames. Set this parameter to 1 to specify all key frames, to 2 to specify every other frame as a key frame, to 3 to specify every third frame as a key frame, and so forth. Your application may change the key frame rate for an active sequence by calling `SetCSequenceKeyFrameRate` (III–1577).

ctable

A handle to a custom **color lookup table**. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the `colorDepth` parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

CompressSequenceBegin

flags

Contains flags (see below) that provide further control information. You must set either `codecFlagUpdatePrevious` or `codecFlagUpdatePreviousComp` to 1. Set unused flags to 0.

desc

A handle that is to receive a formatted `ImageDescription` (IV-2285) structure. The Image Compression Manager resizes this handle for the returned image description structure. Your application should store this image description with the compressed sequence. During the compression operation, the Image Compression Manager and the compressor component update the contents of this image description. Consequently, you should not store the image description until the sequence has been completely processed.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

spatialQuality and temporalQuality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

codec Constants

`anyCodec`

The first compressor or decompressor of the specified type.

`bestSpeedCodec`

The fastest compressor or decompressor of the specified type.

`bestFidelityCodec`

The most accurate compressor or decompressor of the specified type.

bestCompressionCodec

The compressor that produces the smallest resulting data.

flags Constants

codecFlagUpdatePrevious

Controls whether the compressor updates the previous image during compression. This flag is only used with sequences that are being temporally compressed. If you set this flag to 1, the compressor copies the current frame into the previous frame buffer at the end of frame compression.

codecFlagUpdatePreviousComp

Controls whether the compressor updates the previous image buffer with the compressed image. This flag is only used with temporal compression and is similar to the codecFlagUpdatePrevious flag. As with the codecFlagUpdatePrevious flag, if you set this flag to 1, the compressor updates the previous frame buffer at the end of frame compression. However, this flag causes the Image Compression Manager to update the frame buffer using an image obtained by decompressing the results of the most recent compression operation, rather than the source image, which may give better results at the expense of taking more time.

codecFlagWasCompressed

Indicates to the compressor that the image to be compressed has been compressed before. This information may be useful to compressors that can compensate for the image degradation that may otherwise result from repeated compression and decompression of the same image. Set this flag to 1 to indicate that the image was previously compressed. Set this flag to 0 if the image was not previously compressed.

Discussion

The Image Compression Manager prepares for a sequence-compression operation by reserving appropriate system **resources**. Hence you must call `CompressSequenceBegin` before you call `CompressSequenceFrame` (I-124).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V-2792)

Related Java Methods

`quicktime.std.image.CSequence.CSequence()`

CompressSequenceFrame

See Also

You can set or change the previous image buffer for an active sequence, or change its rectangle, by calling `SetCSequencePrev` (III–1579). You can obtain a pointer to a pixel map that was allocated by the compressor by calling `GetCSequencePrevBuffer` (I–406).

CompressSequenceFrame

Compresses one of a sequence of frames.

```
OSErr CompressSequenceFrame (
    ImageSequence          seqID,
    PixMapHandle           src,
    const Rect             *srcRect,
    CodecFlags             flags,
    Ptr                    data,
    long                   *dataSize,
    UInt8                  *similarity,
    ICMCompletionProcRecordPtr asyncCompletionProc );
```

`seqID`

Unique sequence identifier that was returned by `CompressSequenceBegin` (I–119).

`src`

A handle to a pixel map that contains the image to be compressed. The image must be stored in a pixel map structure.

`srcRect`

A pointer to a rectangle defining the portion of the image to compress. The compressor applies this rectangle to the image stored in the buffer referred to by the `src` parameter.

`flags`

Specifies flags (see below) that provide further control information. You must set either `codecFlagUpdatePrevious` or `codecFlagUpdatePreviousComp` to 1. Set unused flags to 0.

`data`

Points to a location to receive the compressed image data. It is your program's responsibility to make sure that this location can receive at least as much data as indicated by the `GetMaxCompressionSize` (I–420) function. The Image Compression Manager places the actual size of the compressed image into the

field referred to by the `dataSize` parameter. This pointer must contain a **32-bit clean** address.

`dataSize`

A pointer to a field that is to receive the size, in bytes, of the compressed image.

`similarity`

A pointer to a field that is to receive a similarity value. The `CompressSequenceFrame` function returns a value that indicates the similarity of the current frame to the previous frame. A value of 0 indicates that the current frame is a **key frame** in the sequence. A value of 255 indicates that the current frame is identical to the previous frame. Values from 1 through 254 indicate relative similarity, ranging from very different (1) to very similar (254).

`asyncCompletionProc`

Points to an `ICMCompletionProc` (III–2087) callback. The compressor calls your completion function when an asynchronous compression operation is complete. You can cause the compression to be performed asynchronously by specifying a completion function if the compressor supports asynchronous compression.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`codecFlagUpdatePrevious`

Controls whether the compressor updates the previous image during compression. This flag is only used with sequences that are being temporally compressed. If you set this flag to 1, the compressor copies the current frame into the previous frame buffer at the end of frame compression.

`codecFlagWasCompressed`

Indicates to the compressor that the image to be compressed has been compressed before. This information may be useful to compressors that can compensate for the image degradation that may otherwise result from repeated compression and decompression of the same image. Set this flag to 1 to indicate that the image was previously compressed. Set this flag to 0 if the image was not previously compressed.

`codecFlagUpdatePreviousComp`

Controls whether the compressor updates the previous image buffer with the compressed image. This flag is only used with temporal compression and is similar to the `codecFlagUpdatePrevious` flag. If you set this flag to 1, the compressor updates the previous frame buffer at the end of frame compression. However, this flag causes the Image Compression Manager to update the frame

CompressSequenceFrame

buffer using an image obtained by decompressing the results of the most recent compression operation, rather than the source image.

`codecFlagForceKeyFrame`

Controls whether the compressor creates a **key frame** from the current image. This flag is only used with temporal compression. If you set this flag to 1, the compressor makes the current image a key frame. If you set this flag to 0, the compressor decides based on other criteria, such as the key frame rate, whether to create a key frame from the current image. If you don't want any key frames other than the ones that are forced, set the key frame rate for the sequence to 0.

`codecFlagLiveGrab`

Indicates to the compressor that speed is of the utmost importance, and that size and quality are of lesser importance. This flag is useful when you are grabbing sequences from a live source where each frame must be compressed quickly.

Discussion

The Image Compression Manager prepares for a sequence-compression operation by reserving appropriate system **resources**. Hence you must call `CompressSequenceBegin` (I-119) before you call `CompressSequenceFrame`.

Special Considerations

If you specify asynchronous operation, you must not read the compressed data until the compressor indicates that the operation is complete by calling your completion function. Set `asyncCompletionProc` to `NIL` to specify synchronous compression. If you set `asyncCompletionProc` to `-1`, the operation is performed asynchronously but the compressor does not call your completion function. If the `asyncCompletionProc` parameter is not `NIL`, the following conditions are in effect: the pixels in the source image must stay valid until the completion function is called with its `codecCompletionSource` flag, and the resulting compressed data is not valid until it is called with its `codecCompletionDest` flag set.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V-2792)

Related Java Methods

`quicktime.std.image.CSequence.compressFrame()`

See Also

See `SCCompressSequenceFrame` (III-1534).

CompShift

Undocumented

```
void CompShift (
    wide    *src,
    short    shift );
```

src

Undocumented

shift

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

CompSquareRoot

Undocumented

```
long CompSquareRoot (
    const wide    *src );
```

src

Undocumented

function result *Undocumented*

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

CompSub

Undocumented

```
void CompSub (
    wide    *src,
    wide    *dst );
```

ConcatMatrix

src

Undocumented

dst

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

ConcatMatrix

Concatenates two matrices, combining the transformations described by both matrices into a single matrix.

```
void ConcatMatrix (  
    const MatrixRecord    *a,  
    MatrixRecord          *b );
```

a

A pointer to the source matrix.

b

A pointer to the destination matrix. The `ConcatMatrix` function performs a matrix multiplication operation on the two matrices and leaves the result in the matrix specified by this parameter.

Discussion

This is a matrix multiplication operation, as a result of which $[B] = [B] \times [A]$. Note that matrix multiplication is not commutative.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Managing Matrices” (V-2764)

Related Java Methods

`quicktime.std.image.Matrix.concat()`

ConvertFileToMovieFile

Converts a file to a movie file and supports a user settings dialog box for import operations.

```
OSErr ConvertFileToMovieFile (
    const FSSpec      *inputFile,
    const FSSpec      *outputFile,
    OSType             creator,
    ScriptCode         scriptTag,
    short              *resID,
    long               flags,
    ComponentInstance   userComp,
    MovieProgressUPP    proc,
    long               refCon );
```

inputFile

A pointer to the file system specification for the file to be converted into a movie file.

outputFile

A pointer to the file specification for the destination movie file.

creator

The creator value for the file if it is a new one.

scriptTag

The script in which the movie file should be converted. Use the Script Manager constant `smSystemScript` to use the system script; use the `smCurrentScript` constant to use the current script. See *Inside Macintosh: Text* (listed in the **bibliography**) for more information about scripts and script tags.

resID

A pointer to a field that is to receive the **resource** ID of the file to be converted. If you don't want to receive the resource ID, set this parameter to `NIL`.

flags

Contains flags (see below) that control movie file conversion and determine whether or not the user settings dialog box appears.

userComp

Indicates a component or component instance of the movie export component you want to perform the conversion. Otherwise, set this parameter to 0 for the Movie Toolbox to choose the appropriate component. If you pass in a component instance, it will be used by `ConvertFileToMovieFile`. This allows you to communicate directly with the component before using this function to

ConvertFileToMovieFile

establish any conversion parameters. If you pass in a component ID, an instance is created and closed within this function.

proc

Points to your progress callback. To remove a movie's **progress function**, set this parameter to NIL. Set this parameter to -1 for the Movie Toolbox to provide a default progress function. See `MovieProgressProc` (III-2105) for the interface your progress callback must support.

refCon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

flags Constants

`createMovieFileDeleteCurFile`

Indicates whether to delete an existing file. If you set this flag to 1, the Movie Toolbox deletes the file (if it exists) before converting the new movie file. If you set this flag to 0 and the file specified by the `fileSpec` parameter already exists, the Movie Toolbox uses the existing file. In this case, the toolbox ensures that the file has both a data and a **resource** fork.

`movieToFileOnlyExport`

If this bit is set and the `showUserSettingsDialog` bit is set, the Save As dialog box restricts the user to those file formats that are supported by movie data export components.

`movieFileSpecValid`

If this bit is set and the `showUserSettingsDialog` bit is set, the name field of the `outputFile` parameter is used as the default name of the exported file in the Save As dialog box.

`showUserSettingsDialog`

Controls whether the user settings dialog box for the specified import operation can appear. Set this flag to 1 to display the user settings dialog box.

Discussion

Use this function to specify an input file and convert it to a movie file. Because some conversions may take a nontrivial amount of time, you can pass a standard movie **progress function** in the `proc` and `refCon` parameters.

Special Considerations

Once you are finished working with a movie, you should release the **resources** used by the movie by calling `DisposeMovie` (I-268).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Functions” (V–2713)

Related Java Methods

`quicktime.io.QTFile.convertToMovieFile()`

See Also

`ConvertMovieToFile` (I–134) takes a specified movie (or a single track within that movie) and converts it into an output file.

ConvertImage

C

Converts the format of a compressed image.

```
OSErr ConvertImage (
    ImageDescriptionHandle  srcDD,
    Ptr                    srcData,
    short                  colorDepth,
    CTabHandle             ctable,
    CodecQ                 accuracy,
    CodecQ                 quality,
    CodecType              cType,
    CodecComponent         codec,
    ImageDescriptionHandle dstDD,
    Ptr                    dstData );
```

`srcDD`

A handle to the `ImageDescription` (IV–2285) structure that describes the compressed image.

`srcData`

Points to the compressed image data. This pointer must contain a **32-bit clean** address.

`colorDepth`

The depth at which the recompressed image is likely to be viewed. Decompressors may use this as an indication of the color or grayscale resolution of the compressed image. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale,

respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the `GetCodecInfo` (I–385) function.

`ctable`

A handle to a custom **color lookup table**. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the `colorDepth` parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source `ImageDescription` (IV–2285) structure.

`accuracy`

A constant (see below) that defines the accuracy desired in the decompressed image. Values for this parameter are on the same scale as compression quality. For a good display of still images, you should specify at least `codecHighQuality`.

`quality`

A constant (see below) that defines the desired compressed image quality.

`cType`

A compressor type; see “Codec Identifiers” (IV–2655).

`codec`

Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special constant (see below). Specifying a component instance may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation.

`dstDD`

A handle that is to receive a formatted `ImageDescription` (IV–2285) structure. The Image Compression Manager resizes this handle for the returned `ImageDescription` (IV–2285) structure. Your application should store this image description with the compressed image data.

`dstData`

Points to a location to receive the compressed image data. It is your program’s responsibility to make sure that this location can receive at least as much data as indicated by `GetMaxCompressionSize` (I–420). The Image Compression

Manager places the actual size of the compressed image into the `dataSize` field of the image description referred to by the `dstDD` parameter. This pointer must contain a 32-bit-clean address.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

accuracy and quality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

codec Constants

`anyCodec`

The first compressor or decompressor of the specified type.

`bestSpeedCodec`

The fastest compressor or decompressor of the specified type.

`bestFidelityCodec`

The most accurate compressor or decompressor of the specified type.

`bestCompressionCodec`

The compressor that produces the smallest resulting data.

Discussion

The action of this function is essentially equivalent to decompressing and recompressing the image. During the decompression operation, the decompressor uses the `srcDD`, `srcData`, and accuracy parameters. During the subsequent compression operation, the compressor uses the `colorDepth`, `ctable`, `cType`, `codec`, `quality`, `dstDD`, and `dstData` parameters.

ConvertMovieToFile

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pixel Maps” (V-2789)

Related Java Methods

`quicktime.std.image.QTImage.convert()`

ConvertMovieToFile

Takes a specified movie (or a single track within that movie) and converts it into a specified file and type, supporting a Save As dialog box.

```
OSErr ConvertMovieToFile (
    Movie          theMovie,
    Track          onlyTrack,
    FSSpec         *outputFile,
    OSType         fileType,
    OSType         creator,
    ScriptCode     scriptTag,
    short          *resID,
    long           flags,
    ComponentInstance userComp );
```

theMovie

The source movie for this conversion operation. Your application obtains this movie identifier from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), and `NewMovieFromHandle` (II-1084).

onlyTrack

The track within the source movie for this conversion operation. To specify all tracks, set the value of this parameter to 0.

outputFile

A pointer to the file specification for the destination file.

fileType

The data type of the destination file for the movie specified in the parameter `theMovie`.

creator

The creator value for the output file if it is a new one.

scriptTag

The script into which the movie should be converted if the output file is a new one. Use the Script Manager constant `smSystemScript` to use the system script; use the `smCurrentScript` constant to use the current script. See *Inside Macintosh: Text* (listed in the **bibliography**) for more information about scripts and script tags.

resID

A pointer to a field that is to receive the **resource** ID of the open movie. If you don't want to receive this information, set the `resID` parameter to `NIL`.

flags

Contains flags (see below) that control whether and how the Save As dialog box appears.

userComp

If you want a particular movie export component to perform the conversion, you may pass the component or an instance of that component in this parameter. Otherwise, set it to 0 to allow the Movie Toolbox to use the appropriate component. If you pass in a component instance, it is used by `ConvertMovieToFile`. This allows you to communicate directly with the component before making this call to establish any conversion parameters. If you pass in a component ID, an instance is created and closed within this call.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

flags Constants**showUserSettingsDialog**

If this bit is set, the Save As dialog box will be displayed to allow the user to choose the type of file to export to, as well as the file name to export to.

movieToFileOnlyExport

If this bit is set and the `showUserSettingsDialog` bit is set, the Save As dialog box restricts the user to those file formats that are supported by movie data export components.

movieFileSpecValid

If this bit is set and the `showUserSettingsDialog` bit is set, the `name` field of the `outputFile` parameter is used as the default name of the exported file in the Save As dialog box.

Discussion

Your application controls whether a Save As dialog box appears by setting the value of the `flags` parameter. The dialog box lets the user specify the file name and type.

ConvertMovieToFile

Supported types include standard QuickTime movies, flattened movies, single-fork flattened movies, and any format that is supported by a movie data export component. The following code snippets show how to call `ConvertMovieToFile` to provide a simple export capability and how to save a sound-only QuickTime movie as a WAV file.

```
// Providing an export capability with ConvertMovieToFile
err = ConvertMovieToFile (theMovie,      /* identifies movie */
                          NIL,           /* all tracks */
                          NIL,           /* no output file */
                          0,             /* no file type */
                          0,             /* no creator */
                          -1,            /* script */
                          NIL,           /* no resource ID */
                          createMovieFileDeleteCurFile |
                              showUserSettingsDialog |
                              movieToFileOnlyExport,
                          0);             /* no specific component */

// Saving a sound-only QuickTime movie as a WAVE file
// See "Discovering QuickTime," page 257
void SndSnip_SaveSoundMovieAsWAVEFile (Movie theMovie)
{
    StandardFileReply  myReply;
    // have the user select the name and location of the new WAVE file
    StandardPutFile("\pSave sound movie file as:",
                   "\pUntitled.wav", &myReply);

    if (!myReply.sfGood)
        return;
    // use the default progress procedure, if any
    SetMovieProgressProc(theMovie, (MovieProgressUPP)-1L, 0);
    // export the movie into a file
    ConvertMovieToFile( theMovie,          // the movie to convert
                       NIL,                // all tracks in the movie
                       &myReply.sfFile,    // the output file
                       kQTFileTypeWave,     // the output file type
                       FOUR_CHAR_CODE('TVOD'), // the output file creator
                       smSystemScript,      // the script
                       NIL,                 // no resource ID
                                           // to be returned
                       0L,                  // no flags
    );
}
```

```

                                NIL);                                // no specific component
    }

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Functions” (V–2713)

Related Java Methods

`quicktime.std.movies.Movie.convertToFile()`

ConvertTime

C

Converts a time obtained from one time base into a time that is relative to another time base.

```

void ConvertTime (
    TimeRecord    *inout,
    TimeBase      newBase );

```

inout

A pointer to a **time structure** that contains the time value to be converted. The `ConvertTime` function replaces the contents of this time structure with the time value relative to the specified time base.

newBase

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (II–1119) function.

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Discussion

This function includes the rate associated with each time value in the conversion; therefore, you should use this function when you want to convert time values. Both time bases must rely on the same time source, and you must specify the time to be converted in a **time structure**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Times” (V–2762)

ConvertTimeScale

Related Java Methods

`quicktime.std.clocks.TimeRecord.convertTime()`

See Also

Use `ConvertTimeScale` (I–138) to convert durations.

ConvertTimeScale

Converts a time from one time scale into a time that is relative to another time scale.

```
void ConvertTimeScale (  
    TimeRecord    *inout,  
    TimeScale     newScale );
```

inout

A pointer to a **time structure** that contains the time value to be converted. `ConvertTimeScale` replaces the contents of this time structure with the time value relative to the specified time scale.

newScale

The time scale for this operation.

function result You can access this function's error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Discussion

This function does not include the rate associated with the time value in the conversion; therefore, you should use this function when you want to convert time durations, but not when converting time values.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Times” (V–2762)

Related Java Methods

`quicktime.std.clocks.TimeRecord.convertTimeScale()`

See Also

Use `ConvertTime` (I–137) to convert time values.

CopyMatrix

Copies the contents of one matrix into another matrix.

```
void CopyMatrix (
    const MatrixRecord *m1,
    MatrixRecord      *m2 );
```

m1

The source matrix for the copy operation.

m2

A pointer to the destination matrix for the copy operation. CopyMatrix copies the values from the matrix specified by the m1 parameter into this matrix.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Managing Matrices” (V–2764)

Related Java Methods

quicktime.std.image.Matrix.copy(), quicktime.std.image.Matrix.clone()

CopyMovieSelection

Creates a new movie that contains the original movie’s current selection.

```
Movie CopyMovieSelection (
    Movie theMovie );
```

theMovie

The source movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (II–1069), NewMovieFromFile (II–1080), and NewMovieFromHandle (II–1084).

function result The new movie.

Discussion

This function creates a new movie from the source movie’s current selection, but does not change the source movie or the selection. If you have assigned a **progress function** to the source movie, the Movie Toolbox calls that progress function during long copy operations.

CopyMovieSettings

Special Considerations

Your application must dispose of the new movie once you are done with it, using `DisposeMovie` (I–268).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Movie Editing Functions” (V–2746)

Related Java Methods

`quicktime.std.movies.Movie.copySelection()`

CopyMovieSettings

Copies many settings from one movie to another, overwriting the destination settings in the process.

```
OSErr CopyMovieSettings (
    Movie    srcMovie,
    Movie    dstMovie );
```

srcMovie

The source movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

dstMovie

The destination movie for this operation. The `CopyMovieSettings` function uses the settings from the source movie, which is specified by the `srcMovie` parameter, to replace the current settings of this movie.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Use this function to copy certain important settings from one movie to another. It copies the preferred rate and volume, source clipping region, matrix information, and user data; it does not copy the movie’s contents. To work with movie contents, you should use segment editing functions.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Low-Level Movie Editing Functions” (V–2749)

Related Java Methods

`quicktime.std.movies.Movie.copySettings()`

See Also

If you want to work with specific movie characteristics, you can use the Movie Toolbox functions that allow you to manipulate movie settings individually. For example, you use `InsertMovieSegment` (II–762) to copy a segment from one movie to another and `InsertEmptyMovieSegment` (II–759) to insert an empty segment into a movie. You delete a segment from a movie by calling `DeleteMovieSegment` (I–250), or change a segment’s duration by calling `ScaleMovieSegment` (III–1528). These functions are described in the discussion section of `SetMovieGWorld` (III–1619).

C

CopyTrackSettings

Copies many settings from one track to another, overwriting the destination settings.

```
OSErr CopyTrackSettings (
    Track    srcTrack,
    Track    dstTrack );
```

`srcTrack`

The source track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

`dstTrack`

The destination track for this operation. The `CopyTrackSettings` function uses the settings from the source track, which you specify with the `srcTrack` parameter, to replace the current settings of this track.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

This function copies matrix information, track volume, the clipping region, user data, matte information, media language, quality, user data, and other media-specific settings (such as sound balance and video graphics mode). It does not copy any alternate group information pertaining to the track. This function does

CountComponentInstances

not copy the track's contents. To work with track contents, you should use segment-editing functions.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Editing Tracks" (V-2750)

Related Java Methods

`quicktime.std.movies.Track.copySettings()`

See Also

If you want to work with specific track characteristics, you can use the Movie Toolbox functions that allow you to manipulate track settings individually. These functions are described in `SetMovieGWorld` (III-1619).

CountComponentInstances

Allows you to determine the number of open connections being managed by a specified component.

```
long CountComponentInstances (  
    Component      aComponent );
```

`aComponent`

A component instance. Your application obtains the component instance from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131).

function result The number of open connections being managed by the specified component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: "Functions Used By Components" (V-2782)

Related Java Methods

`quicktime.std.comp.Component.count()`

CountComponents

Determines the number of registered components that meet a given set of criteria.

```
long CountComponents (
    ComponentDescription    *looking );
```

looking

A `ComponentDescription` (IV–2212) record. Your application specifies the criteria for the component search in the fields of this record. If you set all the fields to 0, the Component Manager returns the total number of QuickTime components registered in the system.

function result The number of registered components that meet the given selection criteria

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Component Functions Used By Applications” (V–2781)

Related Java Methods

`quicktime.std.comp.ComponentDescription.count()`

CountImageDescriptionExtensionType

Counts the number of extensions of a given type in an `ImageDescriptionHandle`.

```
OSErr CountImageDescriptionExtensionType (
    ImageDescriptionHandle    desc,
    long                      idType,
    long                      *count );
```

desc

A handle to the `ImageDescription` (IV–2285) structure with the extensions to be counted.

idType

Indicates the type of extension to be counted in the specified `ImageDescription` (IV–2285) structure. Set the value of this parameter to 0 to match any extension, and return a count of all of the extensions.

CountUserDataType

count

A pointer to an integer that indicates how many extensions of the given type are in the given `ImageDescription` (IV–2285) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

When used with `GetNextImageDescriptionExtensionType` (I–501), this function allows the application to determine the total set of extensions present in the `ImageDescription` (IV–2285) structure designated by `desc`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Image Descriptions” (V–2790)

CountUserDataType

Determines the number of items of a given type in a **user data list**.

```
short CountUserDataType (  
    UserData    theUserData,  
    OSType      udType );
```

theUserData

The **user data list** for this operation. You obtain this list reference by calling the `GetMovieUserData` (I–499), `GetTrackUserData` (I–546), or `GetMediaUserData` (I–456) functions.

udType

The type. The Movie Toolbox determines the number of items of this type in the **user data list**.

function result The number of items of the given type in the **user data list**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

`quicktime.std.movies.media.UserData.countType()`

CreateMovieFile

Creates a movie file, creates an empty movie which references the file, and opens the movie file with write permission.

```
OSErr CreateMovieFile (
    const FSSpec    *fileSpec,
    OSType          creator,
    ScriptCode      scriptTag,
    long            createMovieFileFlags,
    short           *resRefNum,
    Movie           *newmovie );
```

fileSpec

A pointer to the file system specification for the movie file to be created.

creator

The creator value for the new file.

scriptTag

The script in which the movie file should be created. Use the Script Manager constant `smSystemScript` to use the system script; use the `smCurrentScript` constant to use the current script. See *Inside Macintosh: Text* (listed in the **bibliography**) for more information about scripts and script tags.

createMovieFileFlags

Controls movie file creation flags (see below).

resRefNum

A pointer to a field that is to receive the file reference number for the opened movie file. Your application must use this value when calling other Movie Toolbox functions that work with movie files. If you set this parameter to `NIL`, the Movie Toolbox creates the movie file but does not open the file.

newmovie

A pointer to a field that is to receive the identifier of the new movie. `CreateMovieFile` returns the identifier of the new movie. If the function could not create a new movie, it sets this returned value to `NIL`. If you set this parameter to `NIL`, the Movie Toolbox does not create a movie.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

CreateMovieFile

createMovieFileFlags Constants

createMovieFileDontCreateResFile

Use this flag to create a single-fork movie file that will contain the movie in its data fork. The movie file will not contain a **resource** fork. On non-Macintosh systems, no resource file will be created. You must store a movie in the data fork of any file that has been created with this flag. In addition, attention must be paid to the `resRefNum` and `resId` parameters when calling `AddMovieResource` (I–36) or `UpdateMovieResource` (III–1983) on files created with this flag.

createMovieFileDeleteCurFile

Indicates whether to delete an existing file. If you set this flag to 1, the Movie Toolbox deletes the file (if it exists) before creating the new movie file. If you set this flag to 0 and the file specified by the `fileSpec` parameter already exists, the Movie Toolbox uses the existing file. In this case, the toolbox ensures that the file has both a data and a **resource** fork.

createMovieFileDontCreateMovie

Controls whether `CreateMovieFile` creates a new movie in the movie file. If you set this flag to 1, the Movie Toolbox does not create a movie in the new movie file. In this case, the function ignores the `newMovie` parameter. If you set this flag to 0, the Movie Toolbox creates a movie and returns the movie identifier in the field referred to by the `newMovie` parameter.

createMovieFileDontOpenFile

Controls whether `CreateMovieFile` opens the new movie file. If you set this flag to 1, the Movie Toolbox does not open the new movie file. In this case, the function ignores the `resRefNum` parameter. If you set this flag to 0, the Movie Toolbox opens the new movie file and returns its reference number into the field referred to by the `resRefNum` parameter.

newMovieActive

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. A movie that does not have any tracks can still be active. When the Movie Toolbox tries to play the movie, no images are displayed, because there is no movie data. You can make a movie active or inactive by calling `SetMovieActive` (III–1609).

newMovieDontAutoAlternate

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie; you must enable tracks yourself.

Discussion

The following code snippet shows how `CreateMovieFile` may be used to create and open a QuickTime movie file.


```
// CreateMovieFile coding example
// See "Discovering QuickTime," page 243
void CreateMyCoolMovie (void)
{
    StandardFileReply    sfr;
    Movie                movie = NIL;
    FSSpec               fss;
    short                nFileRefNum = 0;
    short                nResID = movieInDataForkResID;

    StandardPutFile("\pEnter movie file name:", "\puntitled.mov", &sfr);
    if (!sfr.sfGood)
        return;

    CreateMovieFile(&sfr.sfFile,
                   FOUR_CHAR_CODE('TVOD'),
                   smCurrentScript,
                   createMovieFileDeleteCurFile |
                   createMovieFileDontCreateResFile,
                   &nFileRefNum,
                   &movie);

    CreateMyVideoTrack(movie); // See "Discovering QuickTime," page 244
    CreateMySoundTrack(movie); // See "Discovering QuickTime," page 250

    AddMovieResource(movie, nFileRefNum, &nResID, NIL);
    if (nFileRefNum != 0)
        CloseMovieFile(nFileRefNum);
    DisposeMovie(movie);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Movie Functions" (V-2713)

Related Java Methods

```
quicktime.io.QTFile.createMovieFile(),
quicktime.std.movies.Movie.createMovieFile()
```

CreatePortAssociation

See Also

Your application can use the functions described in `NewMovieTrack` (II–1092) to place movie data into the new movie file.

CreatePortAssociation

Associates a data structure of type `GrafPort` with an onscreen native window.

```
GrafPtr CreatePortAssociation (  
    void    *theWnd,  
    Ptr     storage,  
    long    flags );
```

theWnd

Native window to associate the `GrafPort` (IV–2273) with. In Windows, this parameter represents the movie `HWND`.

storage

Pointer to optional storage space you allocate for the graphics port. If you pass `NIL`, QuickTime allocates the space for you.

flags

Flags (see below) that determine how events will be handled.

function result A pointer to a `GrafPort` (IV–2273) structure.

flags Constants

`kQTMCHandlePortEvents`

Asks QTML to handle events.

`kQTMMLNoIdleEvents`

Asks QTML not to send idle events. If you set this flag, QuickTime will not pass periodic idle messages to the `WndProc` associated with this window. In this case it is your responsibility to task any movies playing in this window. When this flag is not set, QuickTime makes sure the `WndProc` associated with the onscreen window gets periodic idle messages so your code can in turn idle any movie controllers contained within that window.

Discussion

This function associates a QuickDraw graphics port with an onscreen window. The graphics port provides a drawing context for QuickTime and QuickDraw. In addition, QuickTime hooks the native window, so that any window state changes are reflected in the associated graphics port.

```
// Registering a QuickTime movie window in Windows
```

```

// See "Discovering QuickTime," page 228
LRESULT CALLBACK WndProc
(
    HWND hwnd,           // Handle to window
    UINT iMsg,           // Message type
    WPARAM wParam,        // Message-dependent parameter
    LPARAM lParam)        // Message-dependent parameter
{
    .
    .
    switch (iMsg) {
        case WM_CREATE:
            CreatePortAssociation(hwnd, NIL, 0L);
            break;
        .
        .
    }
}
// end switch (iMsg)
// end WndProc

```

Special Considerations

Before you dispose of the native window, call `DestroyPortAssociation` (I-254).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QTML.h`

CreateShortcutMovieFile

Creates a movie file that just contains a reference to another movie.

```

OSErr CreateShortcutMovieFile (
    const FSSpec    *fileSpec,
    OSType          creator,
    ScriptCode      scriptTag,
    long            createMovieFileFlags,
    Handle          targetDataRef,
    OSType          targetDataRefType );

```

`fileSpec`

A pointer to the file system specification for the movie file to be created.

`creator`

The creator value for the new file.

CreateShortcutMovieFile

`scriptTag`

The script in which the movie file should be created. Use the Script Manager constant `smSystemScript` to use the system script; use the `smCurrentScript` constant to use the current script. See *Inside Macintosh: Text* (listed in the **bibliography**) for more information about scripts and script tags.

`createMovieFileFlags`

Contains movie file creation flags (see below).

`targetDataRef`

A handle to the data referred to by the movie that this function creates.

`targetDataRefType`

The type of the data referred to by the movie that this function creates; see “Data References” (IV–2661).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

createMovieFileFlags Constants

`flattenAddMovieToDataFork`

Causes the movie to be placed in the data fork of the new movie file, as well as in the **resource** fork. You may use this flag to create movie files that are more easily moved to computer systems other than the Macintosh.

`flattenDontInterleaveFlatten`

Allows you to disable the Movie Toolbox’s data storage optimizations. By default, the Movie Toolbox stores movie data in a format that is optimized for playback. Set this flag to 1 to disable these optimizations.

`flattenActiveTracksOnly`

Causes the Movie Toolbox to add only enabled movie tracks to the new movie file. Use `SetTrackEnabled` (III–1658) to enable and disable movie tracks.

`flattenCompressMovieResource`

Compresses the movie **resource** stored in the file’s data fork, but not the movie resource stored in the resource fork on Mac OS systems.

`flattenFSSpecPtrIsDataRefRecordPtr`

Set this flag to 1 if the `FSSpec` pointer is a pointer to a `DataReferenceRecord` (IV–2233) structure. This capability enables you to flatten movies for devices other than file systems.

`flattenForceMovieResourceBeforeMovieData`

Set this flag when you are creating a fast start movie, to put the movie **resource** at the front of the resulting movie file.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.io.QTFile.createShortcutMovieFile()`

CurveAddAtomToVectorStream

Adds an atom to a **vector data stream**.

```
ComponentResult CurveAddAtomToVectorStream (
    ComponentInstance    effect,
    OSType               atomType,
    Size                 atomSize,
    void                 *pAtomData,
    Handle               vectorStream );
```

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

atomType

The type of atom to add to the **vector data stream**.

atomSize

The size of the data for the atom.

pAtomData

A pointer to the data for the atom.

vectorStream

A handle to the **vector data stream** to which to add the atom.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function adds the atom to the end of the specified **vector data stream** and resizes the vector data stream handle.

Version Notes

Introduced in QuickTime 3 or earlier.

CurveAddPathAtomToVectorStream

Programming Info

C interface file: ImageCodec.h

Programming summary: “Vector Codec Component Functions” (V–2899)

CurveAddPathAtomToVectorStream

Adds a path to a **vector data stream**.

```
ComponentResult CurveAddPathAtomToVectorStream (  
    ComponentInstance    effect,  
    Handle               pathData,  
    Handle               vectorStream );
```

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager’s `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

pathData

A handle to the data for the path.

vectorStream

A handle to the **vector data stream** to which to add the path.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function adds the path to the end of the specified **vector data stream** and resizes the vector data stream handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Vector Codec Component Functions” (V–2899)

CurveAddZeroAtomToVectorStream

Adds a `kCurveEndAtom` to a **vector data stream**; this atom marks the end of the vector data stream,

```
ComponentResult CurveAddZeroAtomToVectorStream (  
    ComponentInstance    effect,
```

```
Handle          vectorStream );
```

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

vectorStream

A handle to the **vector data stream** to which to add the `kCurveEndAtom` atom.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function adds a `kCurveEndAtom` atom to the end of the specified **vector data stream** and resizes the vector data stream handle. The `kCurveEndAtom` atom is required at the end of a vector data stream, and there may be only one `kCurveEndAtom` atom in the stream.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Vector Codec Component Functions” (V–2899)

CurveCountPointsInPath

Counts the points along either one of a path's contours or all of its contours.

```
ComponentResult CurveCountPointsInPath (
    ComponentInstance  effect,
    gxPaths            *aPath,
    unsigned long      contourIndex,
    unsigned long      *pCount );
```

effect

The instance of the QuickTime vector codec component for the request.

aPath

A pointer to the path.

contourIndex

The index of the contour to be counted.

pCount

A pointer to a field that is to receive the number of points in the contour or path.

CurveCreateVectorStream

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Vector Codec Component Functions” (V–2899)

CurveCreateVectorStream

Creates a new, empty **vector data stream**.

```
ComponentResult CurveCreateVectorStream (  
    ComponentInstance    effect,  
    Handle               *pStream );
```

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager’s OpenComponent (II–1130) or OpenDefaultComponent (II–1131) function.

pStream

A pointer to the handle that is to receive the newly created **vector data stream**.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The caller is responsible for disposing of the stream when finished with it. This can be done by calling DisposeHandle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Vector Codec Component Functions” (V–2899)

CurveGetAtomDataFromVectorStream

Finds the first atom of a specified type within a **vector data stream** and get its data.

```
ComponentResult CurveGetAtomDataFromVectorStream (  
    ComponentInstance    effect,  
    Handle               vectorStream,
```



```

long          atomType,
long          *dataSize,
Ptr           *dataPtr );

```

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

vectorStream

A handle to the **vector data stream** from which to get the atom.

atomType

The type of atom to find.

dataSize

A pointer to a field that is to receive the size of the atom's data.

dataPtr

A pointer to a pointer to a field that is to receive the atom's data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Before calling this function, your software must lock the handle for the **vector data stream** (with Macintosh, by calling `HLock`). This prevents the handle from being moved, which could invalidate the pointer to the atom data before your software gets the data.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Vector Codec Component Functions” (V–2899)

CurveGetLength

Calculates the length of one of a path's contours or the sum of the lengths of all of its contours.

```

ComponentResult CurveGetLength (
    ComponentInstance  effect,
    gxPaths            *target,
    long               index,
    wide               *wideLength );

```

CurveGetNearestPathPoint

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

target

A pointer to the path.

index

Contains the index of the contour whose length is to be calculated or, if the value is 0, specifies to calculate the lengths of all of the path's contours and return the sum of the lengths.

wideLength

A pointer to a field that is to receive the length.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Vector Codec Component Functions” (V–2899)

CurveGetNearestPathPoint

Finds the closest point on a path to a specified point.

```
ComponentResult CurveGetNearestPathPoint (
    ComponentInstance    effect,
    gxPaths              *aPath,
    FixedPoint            *thePoint,
    unsigned long         *contourIndex,
    unsigned long         *pointIndex,
    Fixed                *theDelta );
```

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

aPath

A pointer to the path.

thePoint

A pointer to a point for which to find the closest point on the path.

contourIndex

A pointer to a field that is to receive the index of the contour that contains the closest point.

pointIndex

A pointer to a field that is to receive the index of the closest point.

theDelta

A pointer to a field that is to receive the distance between the specified point and the closest point in the contour to it.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

In programs where users directly manipulate curves, you can use this function to determine the closest control point to a given point.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Vector Codec Component Functions” (V–2899)

CurveGetPathPoint

Obtains a point from a path and to find out if the point is on the curve.

```
ComponentResult CurveGetPathPoint (
    ComponentInstance    effect,
    gxPaths              *aPath,
    unsigned long        contourIndex,
    unsigned long        pointIndex,
    gxPoint              *thePoint,
    Boolean               *ptIsOnPath );
```

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager’s OpenComponent (II–1130) or OpenDefaultComponent (II–1131) function.

aPath

A pointer to the path.

CurveInsertPointIntoPath

contourIndex

The index of the contour from which to get the point.

pointIndex

The index of the point to get.

thePoint

A pointer to a field that is to receive the point.

ptIsOnPath

A pointer to a field that is to receive a Boolean value that, if TRUE, specifies that the point is on the curve.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function lets programs get a single point from a path without walking the path data.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Vector Codec Component Functions” (V–2899)

See Also

For the corresponding set function, see CurveSetPathPoint (I–162).

CurveInsertPointIntoPath

Adds a new point to a path.

```
ComponentResult CurveInsertPointIntoPath (
    ComponentInstance  effect,
    gxPoint            *aPoint,
    Handle             thePath,
    unsigned long      contourIndex,
    unsigned long      pointIndex,
    Boolean            ptIsOnPath );
```

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager’s OpenComponent (II–1130) or OpenDefaultComponent (II–1131) function.

aPoint

A pointer to the point to add to the path.

thePath

A handle to the path to which to add the point.

contourIndex

The index of the path contour to which to add the point.

pointIndex

The index of the point to add.

ptIsOnPath

If TRUE, specifies that the new point is to be on the path.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function is best for adding a single point to a path rather than large numbers of points.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Vector Codec Component Functions” (V–2899)

CurveLengthToPoint

Obtains the point at a specified distance along a curve.

```
ComponentResult CurveLengthToPoint (
    ComponentInstance effect,
    gxPaths          *target,
    long              index,
    Fixed             length,
    FixedPoint        *location,
    FixedPoint        *tangent );
```

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager’s OpenComponent (II–1130) or OpenDefaultComponent (II–1131) function.

target

A pointer to the path.

CurveNewPath

index

The index of the path contour from which to get the point.

length

The distance along the curve at which to find the point.

location

A pointer to a field that is to receive the point.

tangent

A pointer to a field that is to receive a point that is tangent to the point at the specified distance.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is useful for converting a value to a point, such as when creating an animation that follows a curve.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Vector Codec Component Functions” (V–2899)

CurveNewPath

Creates a new path.

```
ComponentResult CurveNewPath (
    ComponentInstance    effect,
    Handle                *pPath );
```

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager’s `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

pPath

A pointer to a handle that is to receive the new path.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The path created by this function contains one contour and no points. The caller must dispose of the handle when it is finished with it (with Macintosh, by calling `DisposeHandle`).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Vector Codec Component Functions” (V–2899)

CurvePathPointToLength

Obtains the length of a path between specified starting and ending distances that is nearest a point.

```
ComponentResult CurvePathPointToLength (
    ComponentInstance ci,
    gxPaths          *aPath,
    Fixed            startDist,
    Fixed            endDist,
    FixedPoint       *thePoint,
    Fixed            *pLength );
```

ci

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager’s `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

aPath

A pointer to the path.

startDist

The distance along the path at which to start measuring the path’s length.

endDist

The distance along the path at which to stop measuring the path’s length.

thePoint

A pointer to a point; the function measures the path closest to this point.

pLength

A pointer to a field that is to receive the length of the specified part of the path.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

CurveSetPathPoint

Discussion

You can use this function to test if the user has clicked on the specified portion of the curve.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Vector Codec Component Functions” (V–2899)

CurveSetPathPoint

Changes the location of a point in a path.

```
ComponentResult CurveSetPathPoint (
    ComponentInstance    effect,
    gxPaths              *aPath,
    unsigned long        contourIndex,
    unsigned long        pointIndex,
    gxPoint              *thePoint,
    Boolean              ptIsOnPath );
```

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager’s `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

aPath

A pointer to the path.

contourIndex

The index of the path contour that contains the point to change.

pointIndex

The index of the point to change.

thePoint

A pointer to the new value for the point.

ptIsOnPath

If `TRUE`, specifies that the new point is to be on the path.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function edits an existing point location within a path. The function that you call to send the interpolated value to the receiving track is defined as a universal procedure in systems that support the Macintosh Code Fragment Manager (CFM) or is defined as a data procedure for non-CFM systems. With Macintosh, the `TweenerDataUPP` function pointer specifies the function the **tween** component calls with the value generated by the tween operation. A tween component calls this function from its implementation of the `TweenerDoTween` (III–1976) function. You call this function by invoking the function specified in the tween record’s `dataProc` field.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Vector Codec Component Functions” (V–2899)

See Also

For the corresponding get function, see `CurveGetPathPoint` (I–157).

CustomGetFilePreview

Presents an Open dialog box to the user and allows the user to view file **previews**.

```
void CustomGetFilePreview (
    FileFilterYDUPP          fileFilter,
    short                    numTypes,
    ConstSFTYPEListPtr       typeList,
    StandardFileReply        *reply,
    short                    dlgID,
    Point                    where,
    DlgHookYDUPP             dlgHook,
    ModalFilterYDUPP         filterProc,
    ActivationOrderListPtr   activeList,
    ActivateYDUPP            activateProc,
    void                     *yourDataPtr );
```

fileFilter

Points to an optional function that filters the files that are displayed to the user in the dialog box. See `FileFilterYDProc` (III–2083) for details of this function. If you don’t want to supply a filter function, set this parameter to `NIL`. If this parameter is not `NIL`, `CustomGetFilePreview` calls the function for each file to determine whether to display the file to the user. Your function returns a Boolean value that you set to `FALSE` to cause the file to be displayed.

CustomGetFilePreview

CustomGetFilePreview uses this parameter along with the numTypes and typeList parameters to determine which files appear in the dialog box.

numTypes

The number of file types in the array specified by the typeList parameter, which must be a number between 1 and 4. Set this parameter to -1 to display all files.

typeList

Specifies an array of file types to be displayed to the user. The CustomGetFilePreview function only displays files whose type matches an entry in this array (unless you set the numTypes parameter to -1; in this case, the function displays all files to the user).

reply

A pointer to a reply structure that is to receive information about the user's selection. See *Inside Macintosh: Files* (listed in the **bibliography**) for more information about reply structures.

d1gID

The **resource** ID of your custom dialog template. Use this parameter to specify a custom dialog template resource that has a resource type that differs from the standard value. Set this parameter to 0 to use the standard template.

where

The location of the upper-left corner of the dialog box in global coordinates. If you set this point to (-1 -1), the Movie Toolbox centers the dialog box on the main screen. If you set this point to (-2 -2), the Movie Toolbox centers the dialog box on the screen that has the best display characteristics.

d1gHook

Points to a custom dialog function, described in D1gHookYDProc (III-2080). Use this parameter to support a custom dialog box function you have supplied by specifying a dialog template **resource** in your resource file. You specify the dialog template's resource ID with the d1gID parameter. If you are not supplying a custom dialog function, set this parameter to NIL. For more information about using custom dialog functions with the CustomGetFilePreview routine, see *Inside Macintosh: Files* (listed in the **bibliography**).

filterProc

Points to your modal-dialog filter function, described in ModalFilterYDProc (III-2099). This function gives you greater control over the interface presented to the user. See *Inside Macintosh: Files* (listed in the **bibliography**) for more information about using modal-dialog filter functions with CustomGetFile.

activeList

A pointer to a list of all items in the dialog box that can be activated; that is, made the target of keyboard input. The list is stored as an array of integers. The first integer must contain the number of items in the array (not including this count value). The remaining array entries must contain item numbers that specify valid targets of keyboard input, in the order in which the items are to be activated. Set this parameter to NIL to direct all keyboard input to the displayed list of filenames.

activateProc

Points to your activation function, `ActivateYDProc` (III–2075), which controls the highlighting of any items whose shape is known only by your application. See *Inside Macintosh: Files* (listed in the **bibliography**) for more information about standard file activation functions.

yourDataPtr

A pointer to optional data supplied by your application to your callback functions. When the `CustomGetFilePreview` function calls any of your callback functions, it places this data on the stack, making it available to your functions. Set this parameter to NIL if you are not supplying any optional data.

Discussion

This function differs from `StandardGetFilePreview` (III–1910) in that you can provide a custom dialog template and functions to support your template. `CustomGetFilePreview` automatically converts files to movies if your application requests movies. If a file could be converted into a movie file using a movie import component, then the file is shown in the Standard File dialog box.

Special Considerations

`CustomGetFilePreview` does not appear in the interface file `Movies.h`; instead, it's listed in `ImageCompression.h`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Displaying File Previews” (V–2758)

See Also

See *Inside Macintosh: Files* (listed in the **bibliography**) for a description of the `CustomGetFile` routine and for more information about the parameters to `CustomGetFilePreview`.

CutMovieSelection

CutMovieSelection

Creates a new movie that contains the original movie's current selection.

```
Movie CutMovieSelection (  
    Movie    theMovie );
```

theMovie

The source movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result The newly created movie.

Discussion

This function removes the current selection from the original movie and makes the selection into a new movie. After the current selection has been removed from the original movie, the duration of the current selection is 0. The starting time of the current selection is not affected. If you have assigned a **progress function** to the source movie, the Movie Toolbox calls that progress function during long cut operations.

Special Considerations

Your application must dispose of the new movie once you are done with it, using `DisposeMovie` (I–268).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Movie Editing Functions” (V–2746)

Related Java Methods

```
quicktime.std.movies.Movie.cutSelection()
```

D

DataCodecBeginInterruptSafe

Called before performing a compression or decompression operation during interrupt time.

```
ComponentResult DataCodecBeginInterruptSafe (
    DataCodecComponent    dc,
    unsigned long         maxSrcSize );
```

dc

The instance of a compressor or decompressor component for this request. Your software obtains this reference when calling the Component Manager's `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

maxSrcSize

The maximum size of a block of data to be compressed or decompressed, in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function allocates any temporary buffers that are necessary to perform the operation during interrupt time. To release the **resources** used to make the operation safe during interrupt time, call the `DataCodecEndInterruptSafe` (I–172) function or close the instance of the compressor or decompressor component.

Special Considerations

If the function returns an error, your software must not perform compression or decompression operations during interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Data Codec Functions” (V–2807)

DataCodecCompress

Compresses data using the specified compressor component.

```
ComponentResult DataCodecCompress (
    DataCodecComponent    dc,
    void                  *srcData,
    UInt32                 srcSize,
    void                  *dstData,
    UInt32                 dstBufferSize,
    UInt32                 *actualDstSize,
    UInt32                 *decompressSlop );
```

DataCodecCompress

`dc`

The compressor component to used.

`srcData`

A pointer to the data to be compressed.

`srcSize`

The size of the data to be compressed, in bytes.

`dstData`

A pointer to the buffer in which to store the compressed data.

`dstBufferSize`

The size of the buffer in which to store the compressed data, in bytes.

`actualDstSize`

The size of the compressed data that was created, in bytes.

`decompressSlop`

The number of bytes that should be added to the decompression buffer size if decompression occurs in place.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Before calling this function, you should call `DataCodecGetCompressBufferSize` (I–172) to obtain the maximum possible size of the compressed data that will be returned. You can then use this value as the value of the `dstBufferSize` parameter. Note that a buffer for compressed data that is the same size as the uncompressed data may not be large enough: in some cases, the size of the compressed data can be larger than the size of the decompressed data. When you compress data, you should store the size of data before compression at the beginning of the file, immediately before the compressed data. This allows you to obtain the size of the decompressed data and allocate the buffer for storing the decompressed data before calling `DataCodecDecompress` (I–170).

Special Considerations

You can compress **sprites** by calling this function. If you do, you must include the uncompressed size of the sample at the beginning of the sample, before the compressed data, and store the component subtype of the data compressor used to compress the sprite in the `decompressorType` field of the sample’s `SpriteDescription` (IV–2458) structure. You can also compress QuickDraw 3D media samples by calling this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Data Codec Functions” (V-2807)

Related Java Methods

quicktime.std.qtcomponents.DataCodecCompressor.compress()

DataCodecCompressPartial

Undocumented

```
ComponentResult DataCodecCompressPartial (
    DataCodecComponent    dc,
    void                  **next_in,
    unsigned long          *avail_in,
    unsigned long          *total_in,
    void                  **next_out,
    unsigned long          *avail_out,
    unsigned long          *total_out,
    Boolean                tryToFinish,
    Boolean                *didFinish );
```

dc

The compressor component to used.

next_in

Undocumented

avail_in

Undocumented

total_in

Undocumented

next_out

Undocumented

avail_out

Undocumented

total_out

Undocumented

tryToFinish

Undocumented

didFinish

Undocumented

DataCodecDecompress

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

DataCodecDecompress

Decompresses data using the specified compressor component.

```
ComponentResult DataCodecDecompress (  
    DataCodecComponent    dc,  
    void                  *srcData,  
    UInt32                srcSize,  
    void                  *dstData,  
    UInt32                dstBufferSize );
```

dc

The decompressor component to used.

srcData

A pointer to the data to be decompressed.

srcSize

The size of the data to be decompressed, in bytes.

dstData

A pointer to the buffer in which to store the decompressed data.

dstBufferSize

The size of the buffer in which to store the decompressed data, in bytes.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Before allocating the buffer in which to store decompressed data, you need to get the size of the decompressed data. The size is normally stored at the beginning of the file, immediately before the compressed data.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Data Codec Functions” (V–2807)

Related Java Methods

quicktime.std.qtcomponents.DataCodecDecompressor.decompress()

DataCodecDecompressPartial

Undocumented

```
ComponentResult DataCodecDecompressPartial (
    DataCodecComponent dc,
    void **next_in,
    unsigned long *avail_in,
    unsigned long *total_in,
    void **next_out,
    unsigned long *avail_out,
    unsigned long *total_out,
    Boolean *didFinish );
```

dc

The decompressor component to used.

next_in

Undocumented

avail_in

Undocumented

total_in

Undocumented

next_out

Undocumented

avail_out

Undocumented

total_out

Undocumented

didFinish

Undocumented

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

DataCodecEndInterruptSafe

DataCodecEndInterruptSafe

Releases **resources** used by DataCodecBeginInterruptSafe (I–166).

```
ComponentResult DataCodecEndInterruptSafe (  
    DataCodecComponent    dc );
```

dc

The instance of a compressor or decompressor component for this request.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

When your software has finished a compression or decompression operation that must be performed during interrupt time, it can call this function to release any memory of other **resources** that were used by DataCodecBeginInterruptSafe.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Data Codec Functions” (V–2807)

DataCodecGetCompressBufferSize

Returns the maximum possible size of the compressed data that will be returned using the specified compressor component.

```
ComponentResult DataCodecGetCompressBufferSize (  
    DataCodecComponent    dc,  
    UInt32                srcSize,  
    UInt32                *dstSize );
```

dc

The compressor component to used.

srcSize

The size in bytes of the data being compressed.

dstSize

A pointer to the maximum size of the compressed data that will be returned.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The actual size of the compressed data will likely be smaller than the size you initially have to allocate, so after you compress the data you should shrink the compressed data handle down to the actual data size. When compressing a movie, allocate an extra ten 32-bit integers of space to store the compressed movie **resource** header information, as shown in the following code sample:

```
unsigned long compressedSize;
Handle compressedData;

DataCodecGetCompressBufferSize(dataCompressor, movieResourceSize,
&compressedSize);

compressedData = NewHandle(compressedSize + (10 * sizeof(UInt32)));
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Related Java Methods

quicktime.std.qtcomponents.DataCodecCompressor.getCompressBufferSize()

See Also

DataCodecCompress (I–167).

DataHAddMovie

Assigns movie data to a data handler.

```
ComponentResult DataHAddMovie (
    DataHandler    dh,
    Movie          theMovie,
    short          *id );
```

dh

A data handler component.

theMovie

A movie identifier. Your application obtains this identifier from such functions as NewMovie (II–1069), NewMovieFromFile (II–1080), and NewMovieFromHandle (II–1084).

DataHAppend64

id

A pointer to the field that specifies the **resource** containing the movie data that is to be added.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

DataHAppend64

Appends data to the current data reference.

```
ComponentResult DataHAppend64 (  
    DataHandler      dh,  
    void             *data,  
    wide             *fileOffset,  
    unsigned long     size );
```

dh

A data handler component.

data

A pointer to data to be appended.

fileOffset

A pointer to a 64-bit value that represents the offset in the file of data to be appended.

size

The size of the data to be appended.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

DataHCanUseDataRef

Reports whether a data handler can access the data associated with a specified data reference.

```
ComponentResult DataHCanUseDataRef (
    DataHandler    dh,
    Handle         dataRef,
    long           *useFlags );
```

dh

Identifies the calling program's connection to your data handler component.

dataRef

The data reference. This parameter contains a handle to the information that identifies the container in question.

useFlags

A pointer to a field of flags (see below) that your data handler component uses to indicate its ability to access the container identified by the *dataRef* parameter. Set all appropriate flags to 1; set unused flags to 0. For example, if your component supports networked multimedia servers using a special set of protocols, your data handler should set the `kDataHCanRead` and `kDataHCanSpecialRead` flags to 1 for any container that is on that server. In addition, if your component can write to the server, set the `kDataHCanWrite` and `kDataHCanSpecialWrite` flags to 1 (perhaps along with `kDataHCanStreamingWrite`). If your data handler cannot access the container, set the whole field to 0.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

useFlags Constants

`kDataHCanRead`

Indicates that your data handler can read from the container.

`kDataHSpecialRead`

Indicates that your data handler can read from the container using a specialized method. For example, your data handler might support access to networked multimedia servers using a special protocol. In that case, your component would set this flag to 1 whenever the data reference identifies a container on a supported server.

`kDataHSpecialReadFile`

Indicates that your data handler can read from the container using a specialized method that is particular to the type of container in question (for example, a HyperCard stack). This flag represents a special case of the `kDataHSpecialRead`

DataHCloseForRead

flag. That is, this flag is appropriate only if you have also set `kDataHSpecialRead` to 1.

`kDataHCanWrite`

Indicates that your data handler can write data to the container. In particular, use this flag to indicate that your data handler's `DataHPutData` (I-216) function will work with this data reference.

`kDataHSpecialWrite`

Indicates that your data handler can write to the container using a specialized method. As with the `kDataHSpecialRead` flag, your data handler would use this flag to indicate that the data reference identifies a container which your component can access using specialized support (for example, special network protocols).

`kDataHCanStreamingWrite`

Indicates that your data handler can support the special write functions for capturing movie data when writing to this container.

Discussion

Apple's standard data handler sets both the `kDataHCanRead` and `kDataHCanWrite` flags to 1 for any data reference it receives, indicating that it can read from and write to any volume.

Special Considerations

Your data handler may use any facilities necessary to determine whether it can access the container. Bear in mind, though, that your component should try to be as quick about this determination as possible, in order to minimize the chance that the delay will be noticed by the user.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: "Selecting a Data Handler" (V-2838)

See Also

The Movie Toolbox calls your component's `DataHGetVolumeList` (I-207) function to retrieve your data handler's capabilities for an entire volume.

DataHCloseForRead

Closes read-only access to its data reference.

```
ComponentResult DataHCloseForRead (
```

```
DataHandler    dh );
```

dh

Identifies the calling program's connection to your data handler component.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Data handler components provide two basic read facilities: DataHGetData (I–186) function is a fully synchronous read operation, while DataHScheduleData (I–220) is asynchronous. Applications provide scheduling information when they call your component's DataHScheduleData function. When your component processes the queued request, it calls the application's data-handler completion function. Before any application can read data from a data reference, it must open read access to that reference by calling your component's DataHOpenForRead (I–209) function. DataHCloseForRead (I–176) closes that read access path.

Special Considerations

Note that a client program may close its connection to your component (by calling CloseComponent (I–100)) without closing the read path. If this happens, your component should close the data reference before closing the connection.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Reading Movie Data” (V–2839)

See Also

By calling your component's DataHFinishData (I–182) function, applications can force your component to process queued read requests. Applications may call your component's DataHGetScheduleAheadTime (I–206) function in order to determine how far in advance your component prefers to get read requests.

DataHCloseForWrite

Closes write-only access to its data reference.

```
ComponentResult DataHCloseForWrite (
    DataHandler    dh );
```

dh

Identifies the calling program's connection to your data handler component.

DataHCompareDataRef

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Data handlers provide two distinct write facilities: `DataHPutData` (I–216) is a simple synchronous interface that allows applications to append data to the end of a container, and `DataHWrite` is a more capable, asynchronous write function that is suitable for movie capture operations. Before writing data to a data reference, applications must call your component’s `DataHOpenForWrite` (I–210) function to open a write path to the container. `DataHCloseForWrite` closes that write path. As is the case with `DataHScheduleData` (I–220), your component calls the application’s data-handler completion function when you are done with the write request.

Special Considerations

A client program may close its connection to your component (by calling `CloseComponent` (I–100)) without closing the write path. If this happens, your component should close the data reference before closing the connection. Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM “jukebox,” may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Writing Movie Data” (V–2840)

See Also

There are several other helper functions that allow applications to prepare your data handler for a movie capture operation. `DataHCreateFile` (I–180) asks your component to create a new container. The `DataHSetFileSize` (I–227) and `DataHGetFileSize` (I–194) functions work with a container’s size, in bytes. The `DataHGetFreeSpace` (I–198) function allows applications to determine when to make a container larger. `DataHPreextend` (I–214) asks your component to make a container larger. Applications may call your component’s `DataHGetPreferredBlockSize` (I–204) function in order to determine how best to interact with your data handler.

DataHCompareDataRef

Compares a supplied data reference against its current data reference and returns a `Boolean` value indicating whether the data references are equivalent (that is, the two data references identify the same container).


```
ComponentResult DataHCompareDataRef (
    DataHandler    dh,
    Handle         dataRef,
    Boolean        *equal );
```

dh

Identifies the calling program's connection to your data handler component.

dataRef

The data reference to be compared to your component's current data reference. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an **alias**. Apple's memory-based data handler for the Macintosh uses handles (and has a subtype value of 'hdl'), while the **HFS** data handler uses Alias Manager aliases (its subtype value is 'alis'). See "Data References" (IV-2661).

equal

A pointer to a Boolean. Your component should set that Boolean to TRUE if the two data references identify the same container. Otherwise, set the Boolean to FALSE.

function result See "Error Codes" (IV-2663). Returns noErr if there is no error.

Discussion

All data handler components use data references to identify and locate a movie's container. DataHCompareDataRef asks your component to compare a data reference against the current data reference and indicate whether the references are equivalent (that is, refer to the same container). Client programs can correlate data references with data handlers by matching the component's subtype value with the data reference type; the subtype value indicates the type of data reference the component supports. All data handlers with the same subtype value must support the same data reference type.

Special Considerations

Note that your component cannot simply compare the bits in the two data references. For example, two completely different **aliases** may refer to the same **HFS** file. Consequently, you need to completely resolve the data reference in order to determine the file identified by the reference.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: "Identifying Data References" (V-2839)

DataHCreateFile

See Also

`DataHSetDataRef` (I–224) and `DataHGetDataRef` (I–189) allow applications to assign your data handler’s current data reference. `DataHResolveDataRef` (I–218) permits your component to locate a data reference’s container.

DataHCreateFile

Creates a new container that meets the specifications of the current data reference.

```
ComponentResult DataHCreateFile (  
    DataHandler      dh,  
    OSType           creator,  
    Boolean           deleteExisting );
```

dh

Identifies the calling program’s connection to your data handler component.

creator

The creator type of the new container. If the client program sets this parameter to 0, your component should choose a reasonable value (for example, ‘TVOD’, the creator type for Apple’s movie player).

deleteExisting

Indicates whether to delete any existing data. If this parameter is set to `TRUE` and a container already exists for the current data reference, your component should delete that data before creating the new container. If this parameter is set to `FALSE`, your component should preserve any data that resides in the container defined by the current data reference (if there is any).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Data handlers provide two distinct write facilities: `DataHPutData` (I–216) is a simple synchronous interface that allows applications to append data to the end of a container, while `DataHWrite` is a more capable, asynchronous write function that is suitable for movie capture operations. As is the case with `DataHScheduleData` (I–220), your component calls the application’s data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM “jukebox,” may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Writing Movie Data” (V–2840)

See Also

There are several other helper functions that allow applications to prepare your data handler for a movie capture operation. `DataHCreateFile` (I–180) asks your component to create a new container. The `DataHSetFileSize` (I–227) and `DataHGetFileSize` (I–194) functions work with a container’s size, in bytes. The `DataHGetFreeSpace` (I–198) function allows applications to determine when to make a container larger. `DataHPreextend` (I–214) asks your component to make a container larger. Applications may call your component’s `DataHGetPreferredBlockSize` (I–204) function in order to determine how best to interact with your data handler. Before writing data to a data reference, applications must call your component’s `DataHOpenForWrite` (I–210) function to open a write path to the container. `DataHCloseForWrite` (I–177) closes that write path.

D

DataHCreateFileWithFlags

Undocumented

```
ComponentResult DataHCreateFileWithFlags (
    DataHandler    dh,
    OSType         creator,
    Boolean        deleteExisting,
    UInt32         flags );
```

dh

Identifies the calling program’s connection to your data handler component.

creator

The creator type of the new container. If the client program sets this parameter to 0, your component should choose a reasonable value (for example, 'TVOD', the creator type for Apple’s movie player).

deleteExisting

Indicates whether to delete any existing data. If this parameter is set to `TRUE` and a container already exists for the current data reference, your component should delete that data before creating the new container. If this parameter is set to `FALSE`, your component should preserve any data that resides in the container defined by the current data reference (if there is any).

DataHDoesBuffer

flags

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

DataHDoesBuffer

Reports whether a data handler does buffer reads and writes.

```
ComponentResult DataHDoesBuffer (  
    DataHandler    dh,  
    Boolean        *buffersReads,  
    Boolean        *buffersWrites );
```

`dh`

A data handler component.

`buffersReads`

A pointer to a `Boolean` that is returned true if the data handler component does buffer reads, false otherwise.

`buffersWrites`

A pointer to a `Boolean` that is returned true if the data handler component does buffer writes, false otherwise.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

DataHFinishData

Completes or cancels one or more queued read requests.

```
ComponentResult DataHFinishData (  
    DataHandler    dh,
```

```
Ptr          PlaceToPutDataPtr,
Boolean      Cancel );
```

dh

Identifies the calling program's connection to your data handler component.

PlaceToPutDataPtr

The location in memory that is to receive the data. The value of this parameter identifies the specific read request to be completed. If this parameter is set to NIL, the call affects all pending read requests.

Cancel

Indicates whether the calling program wants to cancel the outstanding request. If this parameter is set to TRUE, your data handler should cancel the request (or requests) identified by the *PlaceToPutDataPtr* parameter.

function result See "Error Codes" (IV-2663). Returns *noErr* if there is no error.

Discussion

Client programs use *DataHFinishData* either to cancel outstanding read requests or to demand that the requests be serviced immediately. Note that your component must call the client program's data-handler completion function for each queued request, even though the client program called *DataHFinishData*. Be sure to call the completion function for both canceled and completed read requests.

Special Considerations

Preroll operations are a special case of the immediate service request. The client program will have queued one or more read requests with their scheduled time of delivery set infinitely far into the future. Your data handler queues those requests until the client program calls *DataHFinishData* demanding that all outstanding read requests be satisfied immediately.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: *QuickTimeComponents.h*

Programming summary: "Reading Movie Data" (V-2839)

See Also

Data handler components provide two basic read facilities. *DataHGetData* (I-186) function is a fully synchronous read operation, while *DataHScheduleData* (I-220) is asynchronous. When your component processes the queued request, it calls the application's data-handler completion function. Client programs queue read requests by calling your component's *DataHScheduleData* (I-220) function. Applications may call your component's *DataHGetScheduleAheadTime* (I-206)

DataHFlushCache

function in order to determine how far in advance your component prefers to get read requests. Before any application can read data from a data reference, it must open read access to that reference by calling your component's DataHOpenForRead (I–209) function. DataHCloseForRead (I–176) closes that read access path.

DataHFlushCache

Discards the contents of any cached read buffers.

```
ComponentResult DataHFlushCache (  
    DataHandler    dh );
```

dh

Identifies the calling program's connection to your data handler component.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Note that this function does not invalidate any queued read requests (made by calling your component's DataHScheduleData (I–220) function).

Special Considerations

Client programs may call this function if they have, in some way, changed the container associated with the current data reference on their own. Under these circumstances, data your component may have read and cached in anticipation of future read requests from the client may be invalid.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Managing Data Handler Components” (V–2841)

DataHFlushData

Forces any data in your component's write buffers to be written to the device that contains the current data reference.

```
ComponentResult DataHFlushData (  
    DataHandler    dh );
```

dh

Identifies the calling program's connection to your data handler component.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function is essentially analogous to the Mac OS File Manager's PBFlushFile function. The client program may call this function after any write operation (either DataHPutData (I–216) or DataHWrite (I–232)). Your component should do what is necessary to make sure that the data is written to the storage device that contains the current data reference.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Managing Data Handler Components” (V–2841)

DataHGetAvailableFileSize

Returns the available file size for a data handler component.

```
ComponentResult DataHGetAvailableFileSize (  
    DataHandler  dh,  
    long         *fileSize );
```

dh

A data handler component.

fileSize

A pointer to the file size.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

DataHGetCacheSizeLimit

Returns the cache size limit for a data handler component.

DataHGetData

```
ComponentResult DataHGetCacheSizeLimit (  
    DataHandler    dh,  
    Size           *cacheSizeLimit );
```

dh

A data handler component.

cacheSizeLimit

A pointer to the cache size limit.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

See Also

For the corresponding set function, see DataHSetCacheSizeLimit (I–224).

DataHGetData

Reads data from its current data reference, which is a synchronous read operation.

```
ComponentResult DataHGetData (  
    DataHandler    dh,  
    Handle         h,  
    long           hOffset,  
    long           offset,  
    long           size );
```

dh

Identifies the calling program’s connection to your data handler component.

h

The handle to receive the data.

hOffset

Identifies the offset into the handle where your component should return the data.

offset

The offset in the data reference from which your component is to read.

size

The number of bytes to read.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Data handler components provide two basic read facilities. DataHGetData function is a fully synchronous read operation, while DataHScheduleData (I–220) is asynchronous. (Applications provide scheduling information when they call DataHScheduleData.) Before any application can read data from a data reference, it must open read access to that reference by calling your component’s DataHOpenForRead (I–209) function. DataHCloseForRead (I–176) closes that read access path. When your component processes the queued request, it calls the application’s data-handler completion function.

Special Considerations

Note that the Movie Toolbox may try to read data from a data reference without calling your component’s DataHOpenForRead (I–209) function. If this happens, your component should open the data reference for read-only access, respond to the read request, and then leave the data reference open in anticipation of later read requests.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Reading Movie Data” (V–2839)

See Also

DataHGetData provides a high-level read interface. This is a synchronous read operation; that is, the client program’s execution is blocked until your component returns control from this function. As a result, most time-critical clients use the DataHScheduleData (I–220) function to read data. By calling your component’s DataHFinishData (I–182) function, applications can force your component to process queued read requests. Applications may call your component’s DataHGetScheduleAheadTime (I–206) function in order to determine how far in advance your component prefers to get read requests. Client programs can force your component to invalidate any cached data by calling your component’s DataHFlushCache (I–184) function.

DataHGetDataAvailability

Undocumented

```
ComponentResult DataHGetDataAvailability (
    DataHandler  dh,
    long         offset,
```

DataHGetDataInBuffer

```
long          len,  
long          *missing_offset,  
long          *missing_len );
```

dh

A data handler component.

offset

Undocumented

len

Undocumented

missing_offset

Undocumented

missing_len

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeComponents.h

DataHGetDataInBuffer

Returns the information about the data in a data handler component’s buffer.

```
ComponentResult DataHGetDataInBuffer (  
    DataHandler  dh,  
    long         startOffset,  
    long         *size );
```

dh

A data handler component.

startOffset

The offset to the start of the data.

size

The size of the data.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

DataHGetDataRate

Undocumented

```
ComponentResult DataHGetDataRate (
    DataHandler  dh,
    long         flags,
    long         *bytesPerSecond );
```

dh

A data handler component.

flags

Undocumented

bytesPerSecond

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

DataHGetDataRef

Retrieves your component’s current data reference.

```
ComponentResult DataHGetDataRef (
    DataHandler  dh,
    Handle       *dataRef );
```

dh

Identifies the calling program’s connection to your data handler component.

DataHGetDataRefAsType

`dataRef`

A pointer to a data reference handle. Your component should make a copy of its current data reference in a handle and return that handle in this field. The client program is responsible for disposing of that handle. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an **alias**. Apple's memory-based data handler for the Macintosh uses handles (and has a subtype value of 'hnd1'), while the **HFS** data handler uses Alias Manager aliases (its subtype value is 'alis'). See “Data References” (IV–2661).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

All data handler components use data references to identify and locate a movie's container. Client programs can correlate data references with data handlers by matching the component's subtype value with the data reference type; the subtype value indicates the type of data reference the component supports. All data handlers with the same subtype value must support the same data reference type.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Identifying Data References” (V–2839)

See Also

`DataHCompareDataRef` (I–178) asks your component to compare a data reference against the current data reference and indicate whether the references are equivalent (that is, refer to the same container). `DataHResolveDataRef` (I–218) permits your component to locate a data reference's container. For the set function corresponding to `DataHGetDataRef`, see `DataHSetDataRef` (I–224).

DataHGetDataRefAsType

Retrieves a data handler component's current data reference of a given type.

```
ComponentResult DataHGetDataRefAsType (
    DataHandler    dh,
    OSType         requestedType,
    Handle         *dataRef );
```

dh

A data handler component.

requestedType

The type of the data reference to retrieve; see “Data References” (IV–2661).

dataRef

A pointer to a data reference handle. Your component should make a copy of its current data reference in a handle and return that handle in this field. The client program is responsible for disposing of that handle. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an **alias**. Apple’s memory-based data handler for the Macintosh uses handles (and has a subtype value of 'hndl '), while the **HFS** data handler uses Alias Manager aliases (its subtype value is 'alis '). See “Data References” (IV–2661).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DataHGetDataRefExtension

Retrieves your component’s current data reference extension data.

```
ComponentResult DataHGetDataRefExtension (
    DataHandler    dh,
    Handle          *extension,
    OSType         idType );
```

dh

A data handler component.

extension

A pointer to a handle to the extension data.

idType

A four-byte signature identifying the type of extension data.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

DataHGetDataRefWithAnchor

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding set function, see `DataHSetDataRefExtension` (I–225).

DataHGetDataRefWithAnchor

Retrieves a data handler component’s component’s current data reference and anchor data reference.

```
ComponentResult DataHGetDataRefWithAnchor (
    DataHandler    dh,
    Handle         anchorDataRef,
    OSType         dataRefType,
    Handle         *dataRef );
```

dh

A data handler component.

anchorDataRef

A handle to the anchor data reference.

dataRefType

The type of the data reference; see “Data References” (IV–2661).

dataRef

A pointer to a data reference handle. Your component should make a copy of its current data reference in a handle and return that handle in this field. The client program is responsible for disposing of that handle. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an **alias**. Apple’s memory-based data handler for the Macintosh uses handles (and has a subtype value of `'hndl'`), while the **HFS** data handler uses Alias Manager aliases (its subtype value is `'alis'`). See “Data References” (IV–2661).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding set function, see `DataHSetDataRefWithAnchor` (I–226).

DataHGetDeviceIndex

Returns a value that identifies the device on which a data reference resides.

```
ComponentResult DataHGetDeviceIndex (
    DataHandler  dh,
    long         *deviceIndex );
```

dh

Identifies the calling program’s connection to your data handler component.

deviceIndex

A pointer to a field that your data handler component uses to return a device identifier value. Your component may use any identifier value that is appropriate (for example, Apple’s **HFS** data handler uses the volume reference number). The client program should do nothing with this value other than compare it with other identifiers returned by your data handler component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Some client programs may need to account for the fact that two or more data references reside on the same device. For instance, this may affect storage-allocation requirements. This function allows such client programs to obtain this information from your data handler.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selecting a Data Handler” (V–2838)

DataHGetFileName

Retrieves the name of the file supplying the current data reference for a data handler.

DataHGetFileSize

```
ComponentResult DataHGetFileName (  
    DataHandler    dh,  
    Str255         str );
```

dh

A data handler component.

str

The name of the file as a string.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

DataHGetFileSize

Returns the size, in bytes, of the current data reference.

```
ComponentResult DataHGetFileSize (  
    DataHandler    dh,  
    long           *fileSize );
```

dh

Identifies the calling program’s connection to your data handler component.

fileSize

A pointer to a long integer. Your component returns the size of the container corresponding to the current data reference, in bytes.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function is operationally equivalent to the Mac OS File Manager’s GetEOF function. Before writing data to a data reference, applications must call your component’s DataHOpenForWrite (I–210) function to open a write path to the container. DataHCloseForWrite (I–177) closes that write path. As is the case with DataHScheduleData (I–220), your component calls the application’s data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM “jukebox,” may be read-only devices. As a

result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Writing Movie Data” (V-2840)

See Also

Data handlers provide two distinct write facilities. `DataHPutData` (I-216) is a simple synchronous interface that allows applications to append data to the end of a container. `DataHWrite` is a more capable, asynchronous write function that is suitable for movie capture operations. There are several other helper functions that allow applications to prepare your data handler for a movie capture operation. `DataHCreateFile` (I-180) asks your component to create a new container. The `DataHSetFileSize` (I-227) and `DataHGetFileSize` functions work with a container’s size, in bytes. The `DataHGetFreeSpace` (I-198) function allows applications to determine when to make a container larger. `DataHPreextend` (I-214) asks your component to make a container larger. Applications may call your component’s `DataHGetPreferredBlockSize` (I-204) function in order to determine how best to interact with your data handler. For the set function corresponding to `DataHGetFileSize`, see `DataHSetFileSize` (I-227).

DataHGetFileSize64

Provides a 64-bit version of `DataHGetFileSize` (I-194).

```
ComponentResult DataHGetFileSize64 (
    DataHandler  dh,
    wide         *fileSize );
```

dh

Identifies the calling program’s connection to your data handler component.

fileSize

A pointer to a wide. Your component returns the size of the container corresponding to the current data reference, in bytes.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

DataHGetFileSizeAsync

Discussion

The only difference between this function and `DataHGetFileSize` (I–194) is that the `fileSize` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `DataHGetFileSize` (I–194). For the set function corresponding to `DataHGetFileSize64`, see `DataHSetFileSize64` (I–228).

DataHGetFileSizeAsync

Returns the size of the current data reference, invoking a completion callback.

```
ComponentResult DataHGetFileSizeAsync (
    DataHandler          dh,
    wide                 *fileSize,
    DataHCompletionUPP    completionRtn,
    long                 refCon );
```

`dh`

Identifies the calling program’s connection to your data handler component.

`fileSize`

A pointer to a 64-bit value. Your component returns the size of the container corresponding to the current data reference, in bytes.

`completionRtn`

A pointer to a data handler completion callback, described in `DataHCompletionProc` (III–2078).

`refCon`

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

DataHGetFileTypeOrdering

Returns the preferred ordering for file typing information.

```
ComponentResult DataHGetFileTypeOrdering (
    DataHandler          dh,
    DataHFileTypeOrderingHandle *orderingListHandle );
```

dh

Identifies the calling program's connection to your data handler component.

orderingListHandle

A pointer to a handle to a list of `OSType` values (see below). The list may contain only a subset of the currently defined types (i.e., Mac OS file type, extension, MIME type) to limit the consideration to reasonable types.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

orderingListHandle Constants

`kDataHFileTypeMacOSFileType`

The file type; value is 'ftyp'.

`kDataHFileTypeExtension`

The file type extension; value is 'fext'.

`kDataHFileTypeMIME`

The MIME type; value is 'mime'.

Discussion

If the data handler has not set the data reference, it can choose to return either an error or a reasonable default ordering list.

Special Considerations

Before making a call to this function, the client should have opened the data handler and called `DataHSetDataRef` (I–224) or `DataHSetDataRefWithAnchor` (I–226). This allows the data handler to return a different ordering based on the particular file. This might allow for a data handler to vary its ordering based on the location of the file. For example, on the Mac OS, it might use extensions only on foreign volumes. For other volumes, it might use a Mac OS file type followed by a file extension.

Version Notes

Introduced in QuickTime 5.

DataHGetFreeSpace

Programming Info

C interface file: QuickTimeComponents.h

DataHGetFreeSpace

Reports the number of bytes available on the device that contains the current data reference.

```
ComponentResult DataHGetFreeSpace (  
    DataHandler      dh,  
    unsigned long     *freeSize );
```

dh

Identifies the calling program's connection to your data handler component.

freeSize

A pointer to an unsigned long integer. Your component returns the number of bytes of free space available on the device that contains the container referred to by the current data reference.

function result See "Error Codes" (IV-2663). Returns noErr if there is no error.

Discussion

Before writing data to a data reference, applications must call your component's DataHOpenForWrite (I-210) function to open a write path to the container. DataHCloseForWrite (I-177) closes that write path. As is the case with DataHScheduleData (I-220), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: "Writing Movie Data" (V-2840)

See Also

Data handlers provide two distinct write facilities. DataHPutData (I-216) is a simple synchronous interface that allows applications to append data to the end of a

container. DataHWrite (I–232) is a more capable, asynchronous write function that is suitable for movie capture operations. There are several other helper functions that allow applications to prepare your data handler for a movie capture operation. DataHCreateFile (I–180) asks your component to create a new container. The DataHSetFileSize (I–227) and DataHGetFileSize (I–194) functions work with a container’s size, in bytes. The DataHGetFreeSpace function allows applications to determine when to make a container larger. DataHPreextend (I–214) asks your component to make a container larger. Applications may call your component’s DataHGetPreferredBlockSize (I–204) function in order to determine how best to interact with your data handler.

DataHGetFreeSpace64

Provides a 64-bit version of DataHGetFreeSpace (I–198).

```
ComponentResult DataHGetFreeSpace64 (
    DataHandler    dh,
    wide           *freeSize );
```

dh

A data handler component.

freeSize

A pointer to a 64-bit integer. Your component returns the number of bytes of free space available on the device that contains the container referred to by the current data reference.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The only difference between this function and DataHGetFreeSpace (I–198) is that the freeSize parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeComponents.h

DataHGetInfoFlags

DataHGetInfoFlags

Provides information about the operation of a data handler component.

```
ComponentResult DataHGetInfoFlags (  
    DataHandler    dh,  
    UInt32         *flags );
```

dh

A data handler component.

flags

Flags (see below) that provide information about the data handler.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

flags Constants

kDataHInfoFlagNeverStreams

The data handler doesn’t stream.

kDataHInfoFlagCanUpdateDataRefs

The data handler might update the current data reference.

kDataHInfoFlagNeedsNetworkBandwidth

The data handler may need to occupy the current network.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeComponents.h

DataHGetMacOSFileType

Gets the Mac OS file type for a data handler’s current data reference.

```
ComponentResult DataHGetMacOSFileType (  
    DataHandler    dh,  
    OSType         *fileType );
```

dh

A data handler component.

fileType

A pointer to a file type; see “File Types and Creators” (IV–2668).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding set function, see `DataHSetMacOSFileType` (I-229).

DataHGetMIMEType

Gets the MIME type for a data handler's current data reference.

```
ComponentResult DataHGetMIMEType (
    DataHandler    dh,
    Str255         mimeType );
```

dh

A data handler component.

mimeType

A MIME type string.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

DataHGetMIMETypeAsync

Performs asynchronous discovery of a HTTP/FTP connection's MIME type.

```
ComponentResult DataHGetMIMETypeAsync (
    DataHandler    dh,
    Str255         mimeType,
    DataHCompletionUPP completionRtn,
    long           refCon );
```

dh

A data handler component.

DataHGetMovie

`mimeType`

The MIME type string when (and if) it becomes available.

`completionRtn`

A `DataHCompletionProc` (III–2078) callback that is called when either the data becomes available or there is a failure. Failure can happen if there is a timeout or `DataHFinishData` (I–182) is called with a cancel instruction. The `mimeType` parameter will not be updated until the complete routine executes. If a completion routine is not specified, the call will return immediately.

`refCon`

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

function result If the MIME type is known, this function updates `mimeType` and returns `noErr`. If the information is not known yet, the error `notEnoughDataErr` is returned. This allows non-blocking calls to be made to this function. If it returns another error, that indicates some other failure; see “Error Codes” (IV–2663).

Discussion

This function removes synchronous blocks from QuickTime’s movie opening code.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuicktimeComponents.h`

See Also

`DataHGetMIMeType` (I–201) will block if the MIME type data is not available yet and will continue blocking until either the information becomes available or the operation times out in 60 seconds.

DataHGetMovie

Gets the movie for a data handler’s current data reference.

```
ComponentResult DataHGetMovie (
    DataHandler  dh,
    Movie        *theMovie,
    short        *id );
```

`dh`

A data handler component.

theMovie

A movie identifier. This is the same as the identifier your application obtains from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

id

A pointer to the field that specifies the **resource** containing the movie data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

DataHGetMovieWithFlags

Gets the movie for a data handler’s current data reference, allowing the flags that would be passed to `NewMovieFromDataRef` to be passed to the handler.

```
ComponentResult DataHGetMovieWithFlags (
    DataHandler  dh,
    Movie        *theMovie,
    short        *id,
    short        flags );
```

dh

A data handler component.

theMovie

A movie identifier. This is the same as the identifier your application obtains from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

id

A pointer to the field that specifies the **resource** containing the movie data.

flags

Flags (see below) that control movie characteristics.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

DataHGetPreferredBlockSize

flags Constants

newMovieActive

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. You can make a movie active or inactive by calling `SetMovieActive` (III–1609).

newMovieDontResolveDataRefs

Controls how completely the Movie Toolbox resolves data references in the movie **resource**. If you set this flag to 0, the toolbox tries to completely resolve all data references in the resource. This may involve searching for files on remote volumes. If you set this flag to 1, the toolbox only looks in the specified data reference. If the toolbox cannot completely resolve all the data references, it still returns a valid movie identifier. In this case, the toolbox also sets the current error value to `couldNotResolveDataRef`.

newMovieDontAskUnresolvedDataRefs

Controls whether the Movie Toolbox asks the user to locate files. If you set this flag to 0, the toolbox asks the user to locate files that it cannot find on available volumes. If the toolbox cannot locate a file, even with the user's help, the function returns a valid movie identifier and sets the current error value to `couldNotResolveDataRef`.

newMovieDontAutoAlternate

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie; you must enable and disable tracks yourself.

Discussion

Passing these flags lets you control whether or not the movie returned is active. Additionally, it allows async movie loading to be more efficient.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

`NewMovieFromDataRef` (II–1078).

DataHGetPreferredBlockSize

Reports the block size that it prefers to use when accessing the current data reference.

```
ComponentResult DataHGetPreferredBlockSize (
    DataHandler    dh,
    long           *blockSize );
```

dh

Identifies the calling program's connection to your data handler component.

blockSize

A pointer to a long integer. Your component returns the size of blocks (in bytes) it prefers to use when accessing the current data reference.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

Different devices use different file system block sizes. This function allows your component to report its preferred block size to the client program. Note that the client program is not required to use this block size when making requests. Some clients may, however, try to accommodate your component's preference. Applications may call your component's `DataHGetPreferredBlockSize` function in order to determine how best to interact with your data handler. As is the case with `DataHScheduleData` (I-220), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: "Writing Movie Data" (V-2840)

See Also

Data handlers provide two distinct write facilities. `DataHPutData` (I-216) is a simple synchronous interface that allows applications to append data to the end of a container. `DataHWrite` (I-232) is a more capable, asynchronous write function that is suitable for movie capture operations. There are several other helper functions that allow applications to prepare your data handler for a movie capture operation. `DataHCreateFile` (I-180) asks your component to create a new container. The `DataHSetFileSize` (I-227) and `DataHGetFileSize` (I-194) functions work with a container's size, in bytes. The `DataHGetFreeSpace` (I-198) function allows applications to determine when to make a container larger. `DataHPreextend` (I-214)

DataHGetScheduleAheadTime

asks your component to make a container larger. Before writing data to a data reference, applications must call your component's `DataHOpenForWrite` (I-210) function to open a write path to the container. `DataHCloseForWrite` (I-177) closes that write path.

DataHGetScheduleAheadTime

Reports how far in advance it prefers clients to issue read requests.

```
ComponentResult DataHGetScheduleAheadTime (  
    DataHandler    dh,  
    long           *milliseconds );
```

dh

Identifies the calling program's connection to your data handler component.

milliseconds

A pointer to a long integer. Your component should set this field with a value indicating the number of milliseconds you prefer to receive read requests in advance of the time when the data must be delivered.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This function allows your data handler to tell the client program how far in advance it should schedule its read requests. By default, the Movie Toolbox issues scheduled read requests between 1 and 2 seconds before it needs the data from those requests. For some data handlers, however, this may not be enough time. For example, some data handlers may have to accommodate network delays when processing read requests. Client programs that call `DataHGetScheduleAheadTime` may try to respect your component's preference.

Special Considerations

Note that not all client programs will call `DataHGetScheduleAheadTime`. Further, some clients may not be able to accommodate your preferred time in all cases, even if they have asked for your component's preference. As a result, your component should have a strategy for handling requests that don't provide enough advanced scheduling time. For example, if your component receives a `DataHScheduleData` (I-220) request that it cannot satisfy, it can fail the request with an appropriate error code.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Reading Movie Data” (V–2839)

See Also

Data handler components provide two basic read facilities. `DataHGetData` (I–186) function is a fully synchronous read operation, while `DataHScheduleData` (I–220) is asynchronous. Client programs queue read requests by calling `DataHScheduleData` with scheduling information. When your component processes the queued request, it calls the application’s data-handler completion function. By calling your component’s `DataHFinishData` (I–182) function, applications can force your component to process queued read requests. Before any application can read data from a data reference, it must open read access to that reference by calling your component’s `DataHOpenForRead` (I–209) function. `DataHCloseForRead` (I–176) closes that read access path.

DataHGetVolumeList

Returns a list of the volumes your component can access, along with flags indicating your component’s capabilities for each volume.

```
ComponentResult DataHGetVolumeList (
    DataHandler      dh,
    DataHVolumeList *volumeList );
```

dh

Identifies the calling program’s connection to your data handler component.

volumeList

A pointer to a field that your data handler component uses to return a handle to a volume list. Your component constructs the volume list by allocating a handle and filling it with a series of `DataHVolumeListRecord` (IV–2230) structures, one structure for each volume your component can access.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

To reduce the delay that may result from choosing an appropriate data handler for a volume, the Movie Toolbox maintains a list of data handlers and the volumes they support. The Movie Toolbox uses `DataHGetVolumeList` to build that list. Your data handler may use any facilities necessary to determine whether it can access the volume, including opening a container on the volume. Your component should set to 1 as many of the capability flags as are appropriate for each volume. Don’t

DataHIsStreamingDataHandler

include records for volumes your handler cannot support. For example, if your component supports networked multimedia servers using a special set of protocols, your data handler should set the `kDataHCanRead` and `kDataHCanSpecialRead` flags to 1 for any volume that is on that server. In addition, if your component can write to a volume on the server, set the `kDataHCanWrite` and `kDataHCanSpecialWrite` flags to 1 (perhaps along with `kDataHCanStreamingWrite`). However, your component should create entries only for those volumes that support your protocols.

Special Considerations

It is the calling program's responsibility to dispose of the handle returned by your component. The Movie Toolbox tracks mounting and unmounting removable volumes, and keeps its volume list current. As a result, the Movie Toolbox may call your component's `DataHGetVolumeList` function whenever a removable volume is mounted. If your data handler does not process data that is stored in file system volumes, you need not support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: "Selecting a Data Handler" (V-2838)

DataHIsStreamingDataHandler

Determines if a data handler handles streaming data.

```
ComponentResult DataHIsStreamingDataHandler (
    DataHandler    dh,
    Boolean        *yes );
```

dh

A data handler component.

yes

Pointer to a Boolean that returns TRUE if the data handler handles streaming data, FALSE otherwise.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

DataHOpenForRead

Opens a data handler's current data reference for read-only access.

```
ComponentResult DataHOpenForRead (
    DataHandler    dh );
```

dh

Identifies the calling program's connection to your data handler component.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

After setting your component's current data reference by calling DataHSetDataRef (I–224), client programs call DataHOpenForRead to start reading from the data reference. Your component should open the data reference for read-only access. If the data reference is already open or cannot be opened, return an appropriate error code. As is the case with DataHScheduleData (I–220), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that the Movie Toolbox may try to read data from a data reference without calling your component's DataHOpenForRead function. If this happens, your component should open the data reference for read-only access, respond to the read request, and then leave the data reference open in anticipation of later read requests.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Reading Movie Data” (V–2839)

See Also

Data handler components provide two basic read facilities. DataHGetData (I–186) function is a fully synchronous read operation, while DataHScheduleData (I–220) is asynchronous. By calling your component's DataHFinishData (I–182) function, applications can force your component to process queued read requests. Applications may call your component's DataHGetScheduleAheadTime (I–206) function in order to determine how far in advance your component prefers to get read requests. Before any application can read data from a data reference, it must open read access to that reference by calling your component's DataHOpenForRead function. DataHCloseForRead (I–176) closes that read access path.

DataHOpenForWrite

DataHOpenForWrite

Opens your component's current data reference for write-only access.

```
ComponentResult DataHOpenForWrite (  
    DataHandler    dh );
```

dh

Identifies the calling program's connection to your data handler component.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

After setting your component's current data reference by calling `DataHSetDataRef` (I-224), client programs call `DataHOpenForWrite` to start writing to the data reference. Your component should open the data reference for write-only access. If the data reference is already open or cannot be opened, return an appropriate error code. As is the case with `DataHScheduleData` (I-220), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: "Writing Movie Data" (V-2840)

See Also

Data handlers provide two distinct write facilities. `DataHPutData` (I-216) is a simple synchronous interface that allows applications to append data to the end of a container. `DataHWrite` is a more capable, asynchronous write function that is suitable for movie capture operations. There are several other helper functions that allow applications to prepare your data handler for a movie capture operation. `DataHCreateFile` (I-180) asks your component to create a new container. The `DataHSetFileSize` (I-227) and `DataHGetFileSize` (I-194) functions work with a container's size, in bytes. The `DataHGetFreeSpace` (I-198) function allows applications to determine when to make a container larger. `DataHPreextend` (I-214) asks your component to make a container larger. Applications may call your component's `DataHGetPreferredBlockSize` (I-204) function in order to determine

how best to interact with your data handler. Before writing data to a data reference, applications must call your component's `DataHOpenForWrite` function to open a write path to the container. `DataHCloseForWrite` (I-177) closes that write path.

DataHPlaybackHints

Provides additional information to your component that you may use to optimize the operation of your data handler.

```
ComponentResult DataHPlaybackHints (
    DataHandler      dh,
    long             flags,
    unsigned long     minFileOffset,
    unsigned long     maxFileOffset,
    long             bytesPerSecond );
```

`dh`

Identifies the calling program's connection to your data handler component.

`flags`

Reserved. Set this parameter to 0.

`minFileOffset`

Together with the `maxFileOffset` parameter, specifies the range of data the client program anticipates using from the current data reference. This parameter specifies the offset of earliest byte the program expects to use (that is, the minimum container offset value). If the client expects to access bytes from the beginning of the container, it should set this parameter to 0.

`maxFileOffset`

The offset of the latest byte the program expects to use (that is, the maximum container offset value). If the client expects to use bytes throughout the container, the client should set this parameter to -1.

`bytesPerSecond`

The rate in bytes per second at which your data handler must read data from the data reference in order to keep up with the client program's anticipated needs.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

Applications may call your handler's `DataHPlaybackHints` function to provide you with some guidelines about how those applications plan to use the current data reference. Your component should be prepared to have this function called more

DataHPlaybackHints64

than once for a given data reference. For example, the Movie Toolbox calls this function whenever a movie's playback rate changes. This is a handy way for your data handler to track playback rate changes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: "Managing Data Handler Components" (V-2841)

DataHPlaybackHints64

Provides a 64-bit version of `DataHPlaybackHints` (I-211).

```
ComponentResult DataHPlaybackHints64 (
    DataHandler      dh,
    long             flags,
    const wide       *minFileOffset,
    const wide       *maxFileOffset,
    long             bytesPerSecond );
```

`dh`

Identifies the calling program's connection to your data handler component.

`flags`

Reserved. Set this parameter to 0.

`minFileOffset`

Together with the `maxFileOffset` parameter, specifies the range of data the client program anticipates using from the current data reference. This parameter points to the offset of the earliest byte the program expects to use (that is, the minimum container offset value). If the client expects to access bytes from the beginning of the container, it should set this parameter to 0.

`maxFileOffset`

Pointer to the offset of the latest byte the program expects to use (that is, the maximum container offset value). If the client expects to use bytes throughout the container, the client should set this parameter to -1.

`bytesPerSecond`

The rate in bytes per second at which your data handler must read data from the data reference in order to keep up with the client program's anticipated needs.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

flags Constants**Discussion**

The only difference between this function and `DataHPlaybackHints` (I–211) is that the `minFileOffset` parameter and `maxFileOffset` parameter are 64-bit integers instead of 32-bit integers.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `DataHPlaybackHints` (I–211).

DataHPollRead

Undocumented

```
ComponentResult DataHPollRead (
    DataHandler    dh,
    void           *dataPtr,
    UInt32         *dataSizeSoFar );
```

dh

A data handler component.

dataPtr

Undocumented

dataSizeSoFar

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

DataHPreextend

DataHPreextend

Allocates new space for the current data reference, enlarging the container.

```
ComponentResult DataHPreextend (
    DataHandler      dh,
    unsigned long    maxToAdd,
    unsigned long    *spaceAdded );
```

dh

Identifies the calling program's connection to your data handler component.

maxToAdd

The amount of space to add to the current data reference, in bytes. If the client program sets this parameter to 0, your component should add as much space as it can.

spaceAdded

A pointer to an unsigned long integer. Your component returns the number of bytes it was able to add to the data reference, in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function asks your component to make a container larger, analogous to the Mac OS File Manager's `PBA11ocContig` function. When it is called, your component should allocate contiguous free space. If there is not sufficient contiguous free space to satisfy the request, your component should return a `dskFullErr` error code. Client programs use this function in order to avoid incurring any space-allocation delay when capturing movie data. As is the case with `DataHScheduleData` (I–220), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM “jukebox,” may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Writing Movie Data” (V–2840)

See Also

Data handlers provide two distinct write facilities. `DataHPutData` (I-216) is a simple synchronous interface that allows applications to append data to the end of a container. `DataHWrite` (I-232) is a more capable, asynchronous write function that is suitable for movie capture operations. There are several other helper functions that allow applications to prepare your data handler for a movie capture operation. `DataHCreateFile` (I-180) asks your component to create a new container. The `DataHSetFileSize` (I-227) and `DataHGetFileSize` (I-194) functions work with a container's size, in bytes. The `DataHGetFreeSpace` (I-198) function allows applications to determine when to make a container larger. Applications may call your component's `DataHGetPreferredBlockSize` (I-204) function in order to determine how best to interact with your data handler. Before writing data to a data reference, applications must call your component's `DataHOpenForWrite` (I-210) function to open a write path to the container. `DataHCloseForWrite` (I-177) closes that write path.

DataHPreextend64

Provides a 64-bit version of `DataHPreextend` (I-214).

```
ComponentResult DataHPreextend64 (
    DataHandler    dh,
    const wide     *maxToAdd,
    wide           *spaceAdded );
```

`dh`

Identifies the calling program's connection to your data handler component.

`maxToAdd`

The amount of space to add to the current data reference, in bytes. If the client program sets this parameter to 0, your component should add as much space as it can.

`spaceAdded`

A pointer to a 64-bit integer. Your component returns the number of bytes it was able to add to the data reference, in bytes.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

The only difference between this function and `DataHPreextend` (I-214) is that the `spaceAdded` parameter is a 64-bit integer instead of a 32-bit integer.

DataHPutData

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `DataHPreextend` (I–214).

DataHPutData

Writes data to a component’s current data reference.

```
ComponentResult DataHPutData (
    DataHandler  dh,
    Handle       h,
    long         hOffset,
    long         *offset,
    long         size );
```

dh

Identifies the calling program’s connection to your data handler component.

h

The handle that contains the data to be written to the data reference.

hOffset

Identifies the offset into the handle *h* to the data to be written.

offset

A pointer to a long integer. Your component returns the offset in the data reference at which your component wrote the data.

size

The number of bytes to write.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function provides a high-level write interface. This is a synchronous write operation that only appends data to the end of the current data reference. That is, the client program’s execution is blocked until your component returns control from this function, and the client cannot control where the data is written. As a result, most movie-capture clients (for example, Apple’s sequence grabber

component) use `DataHWrite` (I–232) to write data when creating movies. As is the case with `DataHScheduleData` (I–220), your component calls the application’s data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM “jukebox,” may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Writing Movie Data” (V–2840)

See Also

Data handlers provide two distinct write facilities. `DataHPutData` is a simple synchronous interface that allows applications to append data to the end of a container. `DataHWrite` (I–232) is a more capable, asynchronous write function that is suitable for movie capture operations. There are several other helper functions that allow applications to prepare your data handler for a movie capture operation. `DataHCreateFile` (I–180) asks your component to create a new container. The `DataHSetFileSize` (I–227) and `DataHGetFileSize` (I–194) functions work with a container’s size, in bytes. The `DataHGetFreeSpace` (I–198) function allows applications to determine when to make a container larger. `DataHPreextend` (I–214) asks your component to make a container larger. Applications may call your component’s `DataHGetPreferredBlockSize` (I–204) function in order to determine how best to interact with your data handler. Before writing data to a data reference, applications must call your component’s `DataHOpenForWrite` (I–210) function to open a write path to the container. `DataHCloseForWrite` (I–177) closes that write path. Client programs can force your component to write any cached data by calling your component’s `DataHFlushData` (I–184) function.

DataHReadAsync

Undocumented

```
ComponentResult DataHReadAsync (
    DataHandler      dh,
    void              *dataPtr,
    UInt32            dataSize,
    const wide        *dataOffset,
```

DataHResolveDataRef

```
DataHCompletionUPP    completion,  
long                  refCon );
```

dh

A data handler component.

dataPtr

Undocumented

dataSize

Undocumented

dataOffset

Undocumented

completion

A pointer to a data-handler completion callback, described in DataHCompletionProc (III–2078).

refCon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeComponents.h

DataHResolveDataRef

Locates the container associated with a given data reference.

```
ComponentResult DataHResolveDataRef (  
    DataHandler    dh,  
    Handle         theDataRef,  
    Boolean        *wasChanged,  
    Boolean        userInterfaceAllowed );
```

dh

Identifies the calling program’s connection to your data handler component.

theDataRef

The data reference to be resolved. Different types of containers may require different types of data references. For example, a reference to a memory-based

movie may be a handle, while a reference to a file-based movie may be an **alias**. Apple’s memory-based data handler for the Macintosh uses handles (and has a subtype value of ‘hndl’), while the **HFS** data handler uses Alias Manager aliases (its subtype value is ‘alis’). See “Data References” (IV–2661).

wasChanged

A pointer to a Boolean. Your component should set that Boolean to TRUE if, in locating the container, your data handler updates any information in the data reference.

userInterfaceAllowed

Indicates whether your component may interact with the user when locating the container. If this parameter is set to TRUE, your component may ask the user to help locate the container (for instance, by presenting a Find File dialog box).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function permits your component to locate a data reference’s container. This function is equivalent to the Mac OS Alias Manager’s `ResolveAlias` function. The client program asks your component to locate the container that is associated with a given data reference. If your component determines that the data reference needs to be updated with more accurate location information, it should put the new information in the supplied data reference (and set the Boolean referred to by the `wasChanged` parameter to TRUE). Client programs can correlate data references with data handlers by matching the component’s subtype value with the data reference type; the subtype value indicates the type of data reference the component supports. All data handlers with the same subtype value must support the same data reference type.

Special Considerations

Client programs may call your data handler’s `DataHResolveDataRef` function at any time. Typically, however, the Movie Toolbox uses this function as part of its strategy for opening and reading a movie container. As such, you can expect that the supplied data reference will identify a container that your component can support.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Identifying Data References” (V–2839)

See Also

`DataHSetDataRef` (I–224) and `DataHGetDataRef` (I–189) allow applications to assign your data handler’s current data reference. `DataHCompareDataRef` (I–178) asks your

DataHScheduleData

component to compare a data reference against the current data reference and indicate whether the references are equivalent (that is, refer to the same container).

DataHScheduleData

Reads data from its current data reference, which can be a synchronous read operation or an asynchronous read operation.

```
ComponentResult DataHScheduleData (  
    DataHandler          dh,  
    Ptr                  PlaceToPutDataPtr,  
    long                  FileOffset,  
    long                  DataSize,  
    long                  RefCon,  
    DataHSchedulePtr      scheduleRec,  
    DataHCompletionUPP    CompletionRtn );
```

dh

Identifies the calling program's connection to your data handler component.

PlaceToPutDataPtr

The location in memory that is to receive the data.

FileOffset

The offset in the data reference from which your component is to read.

DataSize

The number of bytes to read.

RefCon

A reference constant that your data handler component should provide to the data-handler completion function specified with the `CompletionRtn` parameter.

scheduleRec

A pointer to a `DataHScheduleRecord` (IV-2229). If this parameter is set to `NIL`, then the client program is requesting a synchronous read operation (that is, your data handler must return the data before returning control to the client program). If this parameter is not set to `NIL`, it must contain the location of a schedule record that has timing information for an asynchronous read request. Your data handler should return control to the client program immediately, and then call the client's data-handler completion function when the data is ready.

CompletionRtn

A pointer to a data handler completion function, described in DataHCompletionProc (III–2078). The client program must provide a completion routine for all asynchronous read requests (that is, all requests that include a valid schedule record). For synchronous requests, client programs should set this parameter to NIL. However, if the function is provided, your handler must call it, even after synchronous requests. When your data handler finishes with the client program’s read request, your component must call this routine even if the request fails. Your component should pass the reference constant that the client program provided with the RefCon parameter.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function provides both a synchronous and an asynchronous read interface. Synchronous read operations work like the DataHGetData (I–186) function; the data handler component returns control to the client program only after it has serviced the read request. Asynchronous read operations allow client programs to schedule read requests in the context of a specified QuickTime time base. Your data handler queues the request and immediately returns control to the calling program. After your component actually reads the data, it calls the client program’s data-handler completion function. If your component cannot satisfy the request (for example, the request requires data more quickly than you can deliver it), your component should reject the request immediately, rather than queuing the request and then calling the client’s data-handler completion function.

```
typedef struct DataHScheduleRecord {
    TimeRecord timeNeededBy; /* schedule info */
    long        extendedID; /* type of data */
    long        extendedVers; /* reserved */
    Fixed       priority; /* priority */
} DataHScheduleRecord, *DataHSchedulePtr;
```

Special Considerations

Note that the Movie Toolbox may try to read data from a data reference without calling your component’s DataHOpenForRead (I–209) function. If this happens, your component should open the data reference for read-only access, respond to the read request, and then leave the data reference open in anticipation of later read requests.

Version Notes

Introduced in QuickTime 3 or earlier.

DataHScheduleData64

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Reading Movie Data” (V-2839)

See Also

Data handler components provide two basic read facilities. `DataHGetData` function is a fully synchronous read operation, while `DataHScheduleData` is asynchronous. When your component processes the queued request, it calls the application’s data-handler completion function. By calling your component’s `DataHFinishData` (I-182) function, applications can force your component to process queued read requests. Applications may call your component’s `DataHGetScheduleAheadTime` (I-206) function in order to determine how far in advance your component prefers to get read requests. Before any application can read data from a data reference, it must open read access to that reference by calling your component’s `DataHOpenForRead` (I-209) function. `DataHCloseForRead` (I-176) closes that read access path. Client programs can force your component to invalidate any cached data by calling your component’s `DataHFlushCache` (I-184) function.

DataHScheduleData64

Provides a 64-bit version of `DataHScheduleData` (I-220).

```
ComponentResult DataHScheduleData64 (
    DataHandler          dh,
    Ptr                  PlaceToPutDataPtr,
    const wide           *FileOffset,
    long                 DataSize,
    long                 RefCon,
    DataHSchedulePtr     scheduleRec,
    DataHCompletionUPP   CompletionRtn );
```

`dh`

Identifies the calling program’s connection to your data handler component.

`PlaceToPutDataPtr`

The location in memory that is to receive the data.

`FileOffset`

A pointer to the offset in the data reference from which your component is to read.

`DataSize`

The number of bytes to read.

RefCon

A reference constant that your data handler component should provide to the data-handler completion function specified with the `CompletionRtn` parameter.

scheduleRec

A pointer to a `DataHScheduleRecord` (IV–2229). If this parameter is set to `NIL`, then the client program is requesting a synchronous read operation (that is, your data handler must return the data before returning control to the client program). If this parameter is not set to `NIL`, it must contain the location of a schedule record that has timing information for an asynchronous read request. Your data handler should return control to the client program immediately, and then call the client’s data-handler completion function when the data is ready.

CompletionRtn

A pointer to a data handler completion function, described in `DataHCompletionProc` (III–2078). The client program must provide a completion routine for all asynchronous read requests (that is, all requests that include a valid schedule record). For synchronous requests, client programs should set this parameter to `NIL`. However, if the function is provided, your handler must call it, even after synchronous requests. When your data handler finishes with the client program’s read request, your component must call this routine even if the request fails. Your component should pass the reference constant that the client program provided with the `RefCon` parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The only difference between this function and `DataHScheduleData` (I–220) is that the `FileOffset` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `DataHScheduleData` (I–220).

DataHSetCacheSizeLimit

DataHSetCacheSizeLimit

Sets the cache size limit for a data handler component.

```
ComponentResult DataHSetCacheSizeLimit (  
    DataHandler    dh,  
    Size           cacheSizeLimit );
```

dh

A data handler component.

cacheSizeLimit

A pointer to the cache size limit.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

See Also

For the corresponding get function, see DataHGetCacheSizeLimit (I–185).

DataHSetDataRef

Assigns a data reference to your data handler component.

```
ComponentResult DataHSetDataRef (  
    DataHandler    dh,  
    Handle         dataRef );
```

dh

Identifies the calling program’s connection to your data handler component.

dataRef

The data reference. This parameter contains a handle to the information that identifies the container in question. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an **alias**. For example, Apple’s memory-based data handler for the Macintosh uses handles (and has a subtype value of 'hndl'), while the **HFS** data handler uses Alias Manager aliases (its subtype value is 'alis'). The client program is

responsible for disposing of the handle, so your component must make a copy of it. See “Data References” (IV–2661).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function allows your application to assign your data handler’s current data reference. All data handler components use data references to identify and locate a movie’s container. Client programs can correlate data references with data handlers by matching the component’s subtype value with the data reference type; the subtype value indicates the type of data reference the component supports. All data handlers with the same subtype value must support the same data reference type.

Special Considerations

Note that the type of data reference always corresponds to the type that your component supports, and that you specify in the component subtype value of your data handler. As a result, the client program does not provide a data reference type value (unlike the Movie Toolbox’s data reference functions).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Identifying Data References” (V–2839)

See Also

`DataHCompareDataRef` (I–178) asks your component to compare a data reference against the current data reference and indicate whether the references are equivalent (that is, refer to the same container). `DataHResolveDataRef` (I–218) permits your component to locate a data reference’s container. For the `get` function corresponding to `DataHSetDataRef`, see `DataHGetDataRef` (I–189).

DataHSetDataRefExtension

Sets your component’s current data reference extension data.

```
ComponentResult DataHSetDataRefExtension (
    DataHandler  dh,
    Handle       extension,
    OSType       idType );
```

`dh`

A data handler component.

DataHSetDataRefWithAnchor

extension

A handle to the extension data.

idType

A four-byte signature identifying the type of extension data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding get function, see `DataHGetDataRefExtension` (I–191).

DataHSetDataRefWithAnchor

Sets the data reference and anchor data reference for a data handler.

```
ComponentResult DataHSetDataRefWithAnchor (
    DataHandler    dh,
    Handle          anchorDataRef,
    OSType          dataRefType,
    Handle          dataRef );
```

dh

A data handler component.

anchorDataRef

A handle to the anchor data reference.

dataRefType

The type of the data reference. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an **alias**. Apple’s memory-based data handler for the Macintosh uses handles (and has a subtype value of `'hndl'`), while the **HFS** data handler uses Alias Manager aliases (its subtype value is `'alis'`). See “Data References” (IV–2661).

dataRef

A data reference handle. Your component should make a copy of its current data reference in a handle and return that handle in this field. The client program is responsible for disposing of that handle.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

See Also

For the corresponding get function, see DataHGetDataRefWithAnchor (I–192).

DataHSetFileSize

Sets the size, in bytes, of the current data reference.

```
ComponentResult DataHSetFileSize (
    DataHandler    dh,
    long           fileSize );
```

dh

Identifies the calling program’s connection to your data handler component.

fileSize

The new size of the container corresponding to the current data reference, in bytes.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function is operationally equivalent to the Mac OS File Manager’s SetEOF function. If the client program specifies a new size that is greater than the current size, your component should extend the container to accommodate that new size. If the client program specifies a container size of 0, your component should free all of the space occupied by the container.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM “jukebox,” may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

DataHSetFileSize64

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Writing Movie Data” (V–2840)

See Also

Data handlers provide two distinct write facilities. `DataHPutData` (I–216) is a simple synchronous interface that allows applications to append data to the end of a container. `DataHWrite` (I–232) is a more capable, asynchronous write function that is suitable for movie capture operations. As is the case with `DataHScheduleData` (I–220), your component calls the application’s data-handler completion function when you are done with the write request. There are several other helper functions that allow applications to prepare your data handler for a movie capture operation.

`DataHCreateFile` (I–180) asks your component to create a new container. The

`DataHGetFileSize` (I–194) function gets a container’s size, in bytes. The

`DataHGetFreeSpace` (I–198) function allows applications to determine when to make a container larger. `DataHPreextend` (I–214) asks your component to make a container larger. Applications may call your component’s `DataHGetPreferredBlockSize`

(I–204) function in order to determine how best to interact with your data handler.

Before writing data to a data reference, applications must call your component’s `DataHOpenForWrite` (I–210) function to open a write path to the container.

`DataHCloseForWrite` (I–177) closes that write path. For the get function

corresponding to `DataHSetFileSize`, see `DataHGetFileSize` (I–194).

DataHSetFileSize64

Provides a 64-bit version of `DataHSetFileSize` (I–227).

```
ComponentResult DataHSetFileSize64 (  
    DataHandler    dh,  
    const wide     *fileSize );
```

dh

Identifies the calling program’s connection to your data handler component.

fileSize

A pointer to the new size of the container corresponding to the current data reference, in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The only difference between this function and `DataHSetFileSize` (I–227) is that the `fileSize` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding get function, see `DataHGetFileSize64` (I–195).

DataHSetMacOSFileType

Sets the Mac OS file type for a data handler’s current data reference.

```
ComponentResult DataHSetMacOSFileType (
    DataHandler  dh,
    OSType       fileType );
```

dh

A data handler component.

fileType

A file type; see “File Types and Creators” (IV–2668).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding get function, see `DataHGetMacOSFileType` (I–200).

DataHSetTimeBase

Sets the time base for a data handler component.

```
ComponentResult DataHSetTimeBase (
    DataHandler  dh,
    TimeBase     tb );
```

DataHSetTimeHints

dh

A data handler component.

tb

A pointer to a TimeBaseRecord (IV–2481) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeComponents.h

DataHSetTimeHints

Undocumented

```
ComponentResult DataHSetTimeHints (
    DataHandler    dh,
    long           flags,
    long           bandwidthPriority,
    TimeScale      scale,
    TimeValue      minTime,
    TimeValue      maxTime );
```

dh

A data handler component.

flags

Undocumented

bandwidthPriority

Undocumented

scale

Undocumented

minTime

Undocumented

maxTime

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

DataHTask

Cedes processor time to your data handler.

```
ComponentResult DataHTask (
    DataHandler    dh );
```

dh

Identifies the calling program's connection to your data handler component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is essentially analogous to the Movie Toolbox's `MoviesTask` (II–973) function. Client programs call this function in order to give your data handler component time to do its work. Because client programs will call this function frequently, and especially so during movie playback or capture, your data handler should return control quickly to the client program.

Special Considerations

Applications should call this function often so that your handler has enough time to do its work.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing Data Handler Components” (V–2841)

DataHUpdateMovie

Updates the movie for a data handler's current data reference.

```
ComponentResult DataHUpdateMovie (
    DataHandler    dh,
    Movie          theMovie,
    short          id );
```

dh

A data handler component.

DataHWrite

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

id

Specifies the **resource** containing the movie data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

DataHWrite

Writes data to its current data reference.

```
ComponentResult DataHWrite (
    DataHandler      dh,
    Ptr              data,
    long             offset,
    long             size,
    DataHCompletionUPP completion,
    long             refCon );
```

dh

Identifies the calling program’s connection to your data handler component.

data

Specifies a pointer to the data to be written. Client programs should lock the memory area holding this data, allowing your component’s `DataHWrite` function to move memory.

offset

The offset (in bytes) to the location in the current data reference at which to write the data.

size

The number of bytes to write.

completion

A pointer to a data-handler completion function, described in `DataHCompletionProc` (III–2078). The client program must provide a completion

routine for all asynchronous write requests. For synchronous requests, client programs should set this parameter to `NIL`. When your data handler finishes with the client program's write request, your component must call this routine even if the request fails. Your component should pass the reference constant that the client program provided with the `refCon` parameter.

`refCon`

A reference constant that your data handler component should provide to the data-handler completion function specified with the `completion` parameter. For synchronous operations, client programs should set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function provides both a synchronous and an asynchronous write interface. Synchronous write operations work like the `DataHPutData` (I–216) function; the data handler component returns control to the client program only after it has serviced the write request. Asynchronous write operations allow client programs to queue write requests. Your data handler queues the request and immediately returns control to the calling program. After your component actually writes the data, it calls the client program's data-handler completion function.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM “jukebox,” may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Writing Movie Data” (V–2840)

See Also

Data handlers provide two distinct write facilities. `DataHPutData` (I–216) is a simple synchronous interface that allows applications to append data to the end of a container. `DataHWrite` is a more capable, asynchronous write function that is suitable for movie capture operations. As is the case with `DataHScheduleData` (I–220), your component calls the application's data-handler completion function when you are done with the write request. There are several other helper functions that allow applications to prepare your data handler for a movie capture operation.

`DataHCreateFile` (I–180) asks your component to create a new container. The `DataHSetFileSize` (I–227) and `DataHGetFileSize` (I–194) functions work with a

DataHWrite64

container's size, in bytes. The `DataHGetFreeSpace` (I–198) function allows applications to determine when to make a container larger. `DataHPreextend` (I–214) asks your component to make a container larger. Applications may call your component's `DataHGetPreferredBlockSize` (I–204) function to determine how best to interact with your data handler. Before writing data to a data reference, applications must call your component's `DataHOpenForWrite` (I–210) function to open a write path to the container. `DataHCloseForWrite` (I–177) closes that write path. Client programs can force your component to write any cached data by calling your component's `DataHFlushData` (I–184) function.

DataHWrite64

Provides a 64-bit version of `DataHWrite` (I–232).

```
ComponentResult DataHWrite64 (  
    DataHandler      dh,  
    Ptr              data,  
    const wide       *offset,  
    long             size,  
    DataHCompletionUPP completion,  
    long             refCon );
```

`dh`

Identifies the calling program's connection to your data handler component.

`data`

Specifies a pointer to the data to be written. Client programs should lock the memory area holding this data, allowing your component's `DataHWrite` function to move memory.

`offset`

A pointer to the offset (in bytes) of the location in the current data reference at which to write the data.

`size`

The number of bytes to write.

`completion`

A pointer to a data-handler completion function, described in `DataHCompletionProc` (III–2078). The client program must provide a completion routine for all asynchronous write requests. For synchronous requests, client programs should set this parameter to `NIL`. When your data handler finishes with the client program's write request, your component must call this routine

even if the request fails. Your component should pass the reference constant that the client program provided with the `refCon` parameter.

`refCon`

A reference constant that your data handler component should provide to the data-handler completion function specified with the `completion` parameter. For synchronous operations, client programs should set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The only difference between this function and `DataHWrite` (I–232) is that the `offset` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `DataHWrite` (I–232).

DecompressImage

Decompresses a single-frame image into a pixel map structure.

```
OSErr DecompressImage (
    Ptr                data,
    ImageDescriptionHandle desc,
    PixMapHandle       dst,
    const Rect         *srcRect,
    const Rect         *dstRect,
    short              mode,
    RgnHandle           mask );
```

`data`

Points to the compressed image data. This pointer must contain a **32-bit clean** address.

`desc`

A handle to the `ImageDescription` (IV–2285) structure that describes the compressed image.

DecompressImage

dst

A handle to the pixel map where the decompressed image is to be displayed. Set the current graphics port to the port that contains this pixel map.

srcRect

A pointer to a rectangle defining the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the compressed image, which is defined by (0,0) and ((**desc).width(**desc).height). If you want to decompress the entire source image, set this parameter to NIL. If the parameter is NIL, the rectangle is set to the rectangle structure of the `ImageDescription` (IV-2285) structure.

dstRect

A pointer to the rectangle into which the decompressed image is to be loaded. The compressor scales the source image to fit into this destination rectangle.

mode

The transfer mode for the operation, as listed in “Graphics Transfer Modes” (IV-2670).

mask

A handle to a clipping region in the destination coordinate system. If specified, the compressor applies this mask to the destination image. If you do not want to mask bits in the destination, set this parameter to NIL.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

Note that `DecompressImage` is invoked through the `StdPix` (III-1912) function. The following code sample illustrates the process of compressing and decompressing a pixel map.

```
// DecompressImage coding example
// See “Discovering QuickTime,” page 286
PicHandle GetQTCompressedPict (PixMapHandle hpmImage)
{
    long                lMaxCompressedSize = 0;
    Handle              hCompressedData = NIL;
    Ptr                 pCompressedData;
    ImageDescriptionHandle hImageDesc = NIL;
    OSErr               nErr;
    PicHandle           hpicPicture = NIL;
    Rect                rectImage = (**hpmImage).bounds;
    CodecType           dwCodecType = kJPEGCodecType;
    CodecComponent       codec = (CodecComponent)anyCodec;
```

```

CodecQ                dwSpatialQuality = codecNormalQuality;
short                 nDepth = 0;           // let ICM choose depth

nErr = GetMaxCompressionSize(hpmImage, &rectImage, nDepth,
                             dwSpatialQuality,
                             dwCodecType,
                             (CompressorComponent)codec,
                             &lMaxCompressedSize);

if (nErr != noErr)
    return NIL;

hImageDesc = (ImageDescriptionHandle)NewHandle(4);
hCompressedData = NewHandle(lMaxCompressedSize);
if ((hCompressedData != NIL) && (hImageDesc != NIL)) {
    MoveHHi(hCompressedData);
    HLock(hCompressedData);
    pCompressedData = StripAddress(*hCompressedData);

    nErr = CompressImage(hpmImage,
                        &rectImage,
                        dwSpatialQuality,
                        dwCodecType,
                        hImageDesc,
                        pCompressedData);

    if (nErr == noErr) {
        ClipRect(&rectImage);
        hpicPicture = OpenPicture(&rectImage);
        nErr = DecompressImage(pCompressedData,
                              hImageDesc,
                              hpmImage,
                              &rectImage,
                              &rectImage,
                              srcCopy,
                              NIL);

        ClosePicture();
    }

    if (theErr || (GetHandleSize((Handle)hpicPicture) ==

```

DecompressSequenceBegin

```
                                sizeof(Picture))) {
        KillPicture(hpicPicture);
        hpicPicture = NIL;
    }
}

if (hImageDesc != NIL)
    DisposeHandle((Handle)hImageDesc);

if (hCompressedData != NIL)
    DisposeHandle(hCompressedData);

return hpicPicture;
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pixel Maps” (V-2789)

Related Java Methods

`quicktime.std.image.QTImage.decompress()`

DecompressSequenceBegin

Obsolete. See `DecompressSequenceBeginS` (I-239).

```
OSErr DecompressSequenceBegin (
    ImageSequence          *seqID,
    ImageDescriptionHandle desc,
    CGrafPtr               port,
    GDHandle                gdh,
    const Rect              *srcRect,
    MatrixRecordPtr         matrix,
    short                   mode,
    RgnHandle               mask,
    CodecFlags              flags,
    CodecQ                  accuracy,
    DecompressorComponent   codec );
```

Programming Info

C interface file: ImageCompression.h

Programming summary: “Working With Sequences” (V-2792)

DecompressSequenceBeginS

Sends a sample image to a decompressor.

```
OSErr DecompressSequenceBeginS (
    ImageSequence          *seqID,
    ImageDescriptionHandle desc,
    Ptr                    data,
    long                   dataSize,
    CGrafPtr               port,
    GDHandle               gdh,
    const Rect              *srcRect,
    MatrixRecordPtr         matrix,
    short                  mode,
    RgnHandle               mask,
    CodecFlags              flags,
    CodecQ                  accuracy,
    DecompressorComponent   codec );
```

seqID

A pointer to a field to receive the unique identifier for the sequence you are creating. You should use this identifier for subsequent calls relating to this decompression sequence.

desc

A handle to the ImageDescription (IV-2285) structure that describes the compressed image.

data

Points to the compressed image data. This pointer must contain a **32-bit clean** address. Ideally, you should pass a pointer to the first frame of the compressed image data, which lets the Image Compression Manager do a better job of preflighting the decompression sequence. If the image data is not available at the time of this call, you can pass NIL for this parameter and 0 for dataSize. If you pass NIL here, then your first call to DecompressSequenceFrameWhen (I-245) may require more setup time.

dataSize

The size of the data buffer, or 0 if you passed NIL in the data parameter.

DecompressSequenceBeginS

`port`

Points to the `CGrafPort` (IV–2168) structure for the destination image.

`gdh`

A handle to the `GDevice` (IV–2264) structure for the destination image. You can pass `NIL` if the `GDevice` is implicit in the port selection (for example, if it is an offscreen graphics world).

`srcRect`

A pointer to a `Rect` (IV–2399) structure that defines the portions of the image to decompress. Pass `NIL` if you want to decompress the entire source image. You can call `SetDSequenceSrcRect` (III–1589) to change the source rectangle for an active decompression sequence.

`matrix`

Points to a `MatrixRecord` (IV–2304) structure that specifies how to transform the image during decompression. Pass `NIL` to use the identity matrix. Your application can change the matrix for an active sequence by calling `SetDSequenceMatrix` (III–1587).

`mode`

The transfer mode for the operation. See “Graphics Transfer Modes” (IV–2670). Your application can change the transfer mode for an active sequence by calling `SetDSequenceTransferMode` (III–1591).

`mask`

A handle to a clipping region in the destination coordinate system. If specified, the compressor applies this mask to the destination image. If you do not want to mask bits in the destination, set this parameter to `NIL`. Your application can change the clipping mask for an active sequence by calling `SetDSequenceMask` (III–1586).

`flags`

Buffer allocation flags (see below).

`accuracy`

A constant (see below) that defines the desired compression accuracy.

`codec`

A decompressor identifier. Specify a particular decompressor by setting this parameter to its identifier. Alternatively, you may use a special identifier (see below). Specifying a component instance may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation.

function result See “Error Codes” (IV–2663). Returns `codecWouldOffscreenErr` if `codecFlagDontUseImageBuffer` is set and the codec requires an

offscreen buffer to decompress to the destination port. Returns `noErr` if there is no error.

flags Constants

`codecFlagUseScreenBuffer`

Obsolete; do not use.

`codecFlagUseImageBuffer`

Controls whether the decompressor allocates an offscreen buffer for decompressing the image. Some decompression sequences can normally go directly to the screen, so passing 1 for this flag will force an offscreen buffer to be created. In other circumstances codecs may require an offscreen buffer, and one will be created regardless of the setting of this flag. You can pass `codecFlagDontUseNewImageBuffer` to prevent an offscreen buffer from being created.

`codecFlagDontUseImageBuffer`

Prevents the Image Compression Manager from creating an offscreen buffer for the decompression sequence. Some codecs require an offscreen buffer to decompress to the destination port successfully; if this is the case and you set this flag, the function returns `codecWouldOffscreenErr`.

accuracy Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

codec Constants

`anyCodec`

The first compressor or decompressor of the specified type.

DecompressSequenceFrame

`bestSpeedCodec`

The fastest compressor or decompressor of the specified type.

`bestFidelityCodec`

The most accurate compressor or decompressor of the specified type.

Discussion

This function lets you pass a compressed sample so a codec can perform preflighting before the first `DecompressSequenceFrameWhen` (I–245) call. To decompress a series of images, call it once to preflight the decompressor, make calls to `DecompressSequenceFrameWhen` to decompress each image in the sequence, then call `CDSequenceEnd` (I–77) when you are done.

Version Notes

Introduced in QuickTime 1.6.1.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V–2792)

Related Java Methods

`quicktime.std.image.DSequence.DSequence()`

See Also

Use `SetDSequenceDataProc` (III–1584) to assign a data-loading function to the sequence. Use `SetDSequenceMatte` (III–1588) to assign a blend matte to the sequence.

DecompressSequenceFrame

Obsolete. See `DecompressSequenceFrameS` (I–243).

```
OSErr DecompressSequenceFrame (
    ImageSequence          seqID,
    Ptr                    data,
    CodecFlags             inFlags,
    CodecFlags             *outFlags,
    ICMCompletionProcRecordPtr  asyncCompletionProc );
```

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V–2792)

DecompressSequenceFrameS

Queues a frame for decompression and specifies the size of the compressed data; new applications should use DecompressSequenceFrameWhen.

```
OSErr DecompressSequenceFrameS (
    ImageSequence          seqID,
    Ptr                    data,
    long                   dataSize,
    CodecFlags             inFlags,
    CodecFlags             *outFlags,
    ICMCompletionProcRecordPtr asyncCompletionProc );
```

seqID

Contains the unique sequence identifier that was returned by the DecompressSequenceBegin (I-238) function.

data

Points to the compressed image data. This pointer must contain a **32-bit clean** address.

dataSize

The size of the data buffer.

inFlags

Contains flags (see below) that provide further control information.

outFlags

Contains status flags (see below). The decompressor updates these flags at the end of the decompression operation.

asyncCompletionProc

Points to an ICMCompletionProcRecord (IV-2279) structure. The compressor calls your completion function when an asynchronous decompression operation is complete. You can cause the decompression to be performed asynchronously by specifying a completion function. If you specify asynchronous operation, you must not read the decompressed image until the decompressor indicates that the operation is complete by calling your completion function. Set asyncCompletionProc to NIL to specify synchronous decompression. If you set asyncCompletionProc to -1, the operation is performed asynchronously but the decompressor does not call your completion function.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

DecompressSequenceFrameS

inFlags Constants

`codecFlagNoScreenUpdate`

Controls whether the decompressor updates the screen image. If you set this flag to 1, the decompressor does not write the current frame to the screen, but does write the frame to its offscreen image buffer (if one was allocated). If you set this flag to 0, the decompressor writes the frame to the screen.

`codecFlagDontOffscreen`

Controls whether the decompressor uses the offscreen buffer during sequence decompression. This flag is only used with sequences that have been temporally compressed. If this flag is set to 1, the decompressor does not use the offscreen buffer during decompression. Instead, the decompressor returns an error. This allows your application to refill the offscreen buffer. If this flag is set to 0, the decompressor uses the offscreen buffer if appropriate.

`codecFlagOnlyScreenUpdate`

Controls whether the decompressor decompresses the current frame. If you set this flag to 1, the decompressor writes the contents of its offscreen image buffer to the screen, but does not decompress the current frame. If you set this flag to 0, the decompressor decompresses the current frame and writes it to the screen. You can set this flag to 1 only if you have allocated an offscreen image buffer for use by the decompressor.

outFlags Constants

`codecFlagUsedNewImageBuffer`

Indicates to your application that the decompressor used the offscreen image buffer for the first time when it processed this frame. If this flag is set to 1, the decompressor used the image buffer for this frame and this is the first time the decompressor used the image buffer in this sequence.

`codecFlagUsedImageBuffer`

Indicates whether the decompressor used the offscreen image buffer. If the decompressor used the image buffer during the decompress operation, it sets this flag to 1. Otherwise, it sets this flag to 0.

`codecFlagDontUseNewImageBuffer`

This input flag forces an error to be returned when a new image buffer would have to be allocated instead of allocating the new buffer.

`codecFlagInterlaceUpdate`

This input flag updates the screen interlacing even and odd scan lines to reduce tearing artifacts (if the decompressor supports this mode).

codecFlagCatchUpDiff

This input flag notifies the codec that the currently displayed frame is being displayed late in an attempt to “catch up” to the current frame, which only happens with compression formats that support frame differencing.

Discussion

This function accepts the same parameters as the DecompressSequenceFrame (I-242) function, with the addition of the dataSize parameter.

Special Considerations

New applications should use DecompressSequenceFrameWhen.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Working With Sequences” (V-2792)

Related Java Methods

quicktime.std.image.DSequence.decompressFrameS()

D

DecompressSequenceFrameWhen

Queues a frame for decompression and specifies the time at which decompression will begin.

```
OSErr DecompressSequenceFrameWhen (
    ImageSequence      seqID,
    Ptr                data,
    long               dataSize,
    CodecFlags         inFlags,
    CodecFlags         *outFlags,
    ICMCompletionProcRecordPtr asyncCompletionProc,
    const ICMFrameTimeRecord *frameTime );
```

seqID

Contains the unique sequence identifier that was returned by the DecompressSequenceBegin (I-238) function.

data

Points to the compressed image data. This pointer must contain a **32-bit clean** address.

dataSize

The size of the data buffer.

DecompressSequenceFrameWhen

`inFlags`

Contains flags (see below) that provide further control information.

`outFlags`

Contains status flags (see below). The decompressor updates these flags at the end of the decompression operation.

`asyncCompletionProc`

Points to an `ICMCompletionProcRecord` (IV–2279) structure. The compressor calls your completion function when an asynchronous decompression operation is complete. You can cause the decompression to be performed asynchronously by specifying a completion function. If you specify asynchronous operation, you must not read the decompressed image until the decompressor indicates that the operation is complete by calling your completion function. Set `asyncCompletionProc` to `NIL` to specify synchronous decompression. If you set `asyncCompletionProc` to `-1`, the operation is performed asynchronously but the decompressor does not call your completion function.

`frameTime`

Points to an `ICMFrameTimeRecord` (IV–2281) structure, which contains the frame’s time information, including the time at which the frame should be displayed, its duration, and the movie’s playback rate. This parameter can be `NIL`, in which case the decompression operation will happen immediately.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inFlags Constants

`codecFlagNoScreenUpdate`

Controls whether the decompressor updates the screen image. If you set this flag to 1, the decompressor does not write the current frame to the screen, but does write the frame to its offscreen image buffer (if one was allocated). If you set this flag to 0, the decompressor writes the frame to the screen.

`codecFlagDontOffscreen`

Controls whether the decompressor uses the offscreen buffer during sequence decompression. This flag is only used with sequences that have been temporally compressed. If this flag is set to 1, the decompressor does not use the offscreen buffer during decompression. Instead, the decompressor returns an error. This allows your application to refill the offscreen buffer. If this flag is set to 0, the decompressor uses the offscreen buffer if appropriate.

`codecFlagOnlyScreenUpdate`

Controls whether the decompressor decompresses the current frame. If you set this flag to 1, the decompressor writes the contents of its offscreen image buffer to the screen, but does not decompress the current frame. If you set this flag to

0, the decompressor decompresses the current frame and writes it to the screen. You can set this flag to 1 only if you have allocated an offscreen image buffer for use by the decompressor.

outFlags Constants

`codecFlagUsedNewImageBuffer`

Indicates to your application that the decompressor used the offscreen image buffer for the first time when it processed this frame. If this flag is set to 1, the decompressor used the image buffer for this frame and this is the first time the decompressor used the image buffer in this sequence.

`codecFlagUsedImageBuffer`

Indicates whether the decompressor used the offscreen image buffer. If the decompressor used the image buffer during the decompress operation, it sets this flag to 1. Otherwise, it sets this flag to 0.

`codecFlagDontUseNewImageBuffer`

This input flag forces an error to be returned when a new image buffer would have to be allocated instead of allocating the new buffer.

`codecFlagInterlaceUpdate`

This input flag updates the screen interlacing even and odd scan lines to reduce tearing artifacts (if the decompressor supports this mode).

`codecFlagCatchUpDiff`

This input flag notifies the codec that the currently displayed frame is being displayed late in an attempt to “catch up” to the current frame, which only happens with compression formats that support frame differencing.

Discussion

The following code snippet shows this function being used to execute one frame of a visual effect.

```
// DecompressSequenceFrameWhen coding example
// See “Discovering QuickTime,” page 310
// Decompress a single step of the effect sequence.
OSErr RunEffect(TimeValue lTime, int nNumberOfSteps)
{
    OSErr          nErr = noErr;
    ICMFrameTimeRecord ftr;

    // Set the timebase time to the step of the sequence to be rendered
    SetTimeBaseValue(timeBase, lTime, nNumberOfSteps);
    ftr.value.lo      = lTime;
    ftr.value.hi      = 0;
```

DecompressSequenceFrameWhen

```
ftr.scale                = nNumberOfSteps;
ftr.base                 = 0;
ftr.duration             = nNumberOfSteps;
ftr.rate                 = 0;
ftr.recordSize           = sizeof(ftr);
ftr.frameNumber          = 1;
ftr.flags                = icmFrameTimeHasVirtualStartTimeAndDuration;
ftr.virtualStartTime.lo  = 0;
ftr.virtualStartTime.hi  = 0;
ftr.virtualDuration      = nNumberOfSteps;

HLock(hEffectDesc);
DecompressSequenceFrameWhen(gEffectSequenceID,
                           StripAddress(*hEffectDesc),
                           GetHandleSize(hEffectDesc),
                           0,
                           0,
                           NIL,
                           &ftr);

HUnlock(hEffectDesc);
}
```

Special Considerations

If the current decompressor component does not support scheduled asynchronous decompression, the Image Compression Manager returns an error code of `codecCantWhenErr`. In this case, the application will need to reissue the request with the `frameTime` parameter set to `NIL`. If the decompressor cannot service your request at a particular time (for example, if its queue is full), the Image Compression Manager returns an error code of `codecCantQueueErr`. The best way to determine whether a decompressor component supports this function is to call the function and test the result code. A decompressor's ability to honor the request may change based on screen depth, clipping settings, and so on.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V-2792)

Related Java Methods

`quicktime.std.image.DSequence.decompressFrameWhen()`

See Also

See DecompressSequenceFrames (I–243).

DelegateComponentCall

Provides an efficient mechanism for passing on requests to a specified component.

```
long DelegateComponentCall (
    ComponentParameters    *originalParams,
    ComponentInstance      ci );
```

originalParams

The ComponentParameters (IV–2216) structure provided to the calling component by the Component Manager.

ci

A component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

function result The component result returned by the specified component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Components.h

Programming summary: “Functions Used By Components” (V–2782)

DeleteMovieFile

Deletes a movie file.

```
OSErr DeleteMovieFile (
    const FSSpec    *fileSpec );
```

fileSpec

A pointer to the file system specification for the movie file to be deleted.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

DeleteMovieSegment

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Functions” (V–2713)

Related Java Methods

`quicktime.io.QTFile.delete()`

DeleteMovieSegment

Removes a specified segment from a movie.

```
OSErr DeleteMovieSegment (
    Movie          theMovie,
    TimeValue      startTime,
    TimeValue      duration );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

startTime

A time value specifying the starting point of the segment to be deleted.

duration

A time value that specifies the duration of the segment to be deleted.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You identify the segment to remove by specifying its starting time and duration. The following code snippet shows `DeleteMovieSegment` being used while adding a modifier track to a movie.

```
// DeleteMovieSegment coding example
// See “Discovering QuickTime,” page 363
Movie          movie1;
TimeValue      101dDuration;
Movie          movie2;
long           lIndex, lOrigTrackCount, lReferenceIndex;
Track          track, trackSprite;
```



```
// get the first track in original movie and position at the start
trackSprite = GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);

// remove all tracks except video in modifier movie
for (lIndex = 1; lIndex <= GetMovieTrackCount(movie2); lIndex++) {
    Track          track = GetMovieIndTrack(movie2, lIndex);
    OSType          dwType;

    GetMediaHandlerDescription(GetTrackMedia(track),
                               &dwType, NIL, NIL);
    if (dwType != VideoMediaType) {
        DisposeMovieTrack(track);
        lIndex--;
    }
}

// add the modifier track to original movie
lOldDuration = GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);

// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, lOldDuration,
                   GetMovieDuration(movie1) - lOldDuration);

// associate the modifier track with the original sprite track
track = GetMovieIndTrack(movie1, lOrigTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                  &lReferenceIndex);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Low-Level Movie Editing Functions” (V-2749)

Related Java Methods

`quicktime.std.movies.Movie.deleteSegment()`

DeleteTrackReference

DeleteTrackReference

Removes a track reference from a track.

```
OSErr DeleteTrackReference (  
    Track      theTrack,  
    OSType     refType,  
    long       index );
```

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

refType

The type of reference.

index

The index value of the reference to be deleted. You obtain this index value when you create the track reference.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

This function deletes a track reference from a track. If there are additional track references with higher index values, the toolbox automatically renumbers those references, decrementing their index values by 1.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Track References” (V–2737)

Related Java Methods

`quicktime.std.movies.Track.deleteReference()`

DeleteTrackSegment

Removes a specified segment from a track.

```
OSErr DeleteTrackSegment (  
    Track      theTrack,
```

```
TimeValue    startTime,
TimeValue    duration );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

startTime

A time value specifying the starting point of the segment to be deleted. This time value must be expressed in the time scale of the movie that contains the source track.

duration

A time value that specifies the duration of the segment to be deleted. This time value must be expressed in the time scale of the movie that contains the source track.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You identify the segment to remove by specifying its starting time and duration.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Tracks” (V–2750)

Related Java Methods

`quicktime.std.movies.Track.deleteSegment()`

Dequeue

Removes a queue element from a Windows `QHdr` (IV–2345) structure.

```
OSErr Dequeue (
    QElemPtr    qElement,
    QHdrPtr     qHeader );
```

qElement

A pointer to a `QElem` (IV–2344) structure.

DestroyPortAssociation

qHeader

A pointer to a QHdr (IV–2345) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Present in QuickTime 3 and later releases.

Programming Info

C interface file: OSUtils.h

DestroyPortAssociation

Removes the graphics port associated with an onscreen window.

```
void DestroyPortAssociation (
    CGrafPtr    cgp );
```

cgp

A pointer to the CGrafPort (IV–2168) structure associated with a native window.

Discussion

This function removes the graphics port associated with an onscreen native window. This association was established previously via the CreatePortAssociation (I–148) call. The DestroyPortAssociation function unhooks the native window’s WndProc and deallocates and window storage, as shown below. You must call this function before you destroy the native window.

```
// DestroyPortAssociation coding example
// See “Discovering QuickTime,” page 229
LRESULT CALLBACK WndProc
    (HWND hwnd,                // Handle to window
     UINT iMsg,                // Message type
     WPARAM wParam,            // Message-dependent parameter
     LPARAM lParam)            // Message-dependent parameter
{
    .
    .
    switch (iMsg) {
        case WM_CLOSE:
            DestroyPortAssociation(GetHwndPort(hwnd));
            break;
```

```

        .
        .
    } // end switch (iMsg)

} // end WndProc

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

DisposeActionsUPP

Disposes of an ActionsUPP pointer.

```
void DisposeActionsUPP (
    ActionsUPP    userUPP );
```

userUPP

An ActionsUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

DisposeAllSprites

Disposes of all **sprites** associated with a sprite world.

```
void DisposeAllSprites (
    SpriteWorld    theSpriteWorld );
```

theSpriteWorld

The **sprite** world for this operation.

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

DisposeCallback

Discussion

This function calls `DisposeSprite` (I–296) for each **sprite** associated with the sprite world.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working with Sprite Worlds” (V–2900)

DisposeCallback

Disposes of a **callback event**.

```
void DisposeCallback (
    QTCallback      cb );
```

`cb`

The **callback event** for the operation. You obtain this value from `NewCallback` (II–1053).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Discussion

You should call this function when you are done with each **callback event**.

Special Considerations

Don’t call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Time Base Callback Functions” (V–2763)

DisposeCharDataHandlerUPP

Disposes of a `CharDataHandlerUPP` pointer.

```
void DisposeCharDataHandlerUPP (
    CharDataHandlerUPP  userUPP );
```

userUPP
A CharDataHandlerUPP pointer.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

DisposeCodecNameList

Disposes of the compressor name list structure you obtained by calling `GetCodecNameList` (I–386).

```
OSErr DisposeCodecNameList (  
    CodecNameSpecListPtr    list );
```

list

Points to the compressor name list to be disposed of. You obtain the compressor list by calling `GetCodecNameList` (I–386).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting Information About Compressor Components” (V–2788)

DisposeCommentHandlerUPP

Disposes of a `CommentHandlerUPP` pointer.

```
void DisposeCommentHandlerUPP (  
    CommentHandlerUPP    userUPP );
```

userUPP

A `CommentHandlerUPP` pointer.

Version Notes

Introduced in QuickTime 5.

DisposeComponentFunctionUPP

Programming Info

C interface file: QuickTimeComponents.h

DisposeComponentFunctionUPP

Disposes of a ComponentFunctionUPP pointer.

```
void DisposeComponentFunctionUPP (  
    ComponentFunctionUPP    userUPP );
```

userUPP

A ComponentFunctionUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Components.h

DisposeComponentMPWorkFunctionUPP

Disposes of a ComponentMPWorkFunctionUPP pointer.

```
void DisposeComponentMPWorkFunctionUPP (  
    ComponentMPWorkFunctionUPP    userUPP );
```

userUPP

A ComponentMPWorkFunctionUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Components.h

DisposeComponentRoutineUPP

Disposes of a ComponentRoutineUPP pointer.

```
void DisposeComponentRoutineUPP (  
    ComponentRoutineUPP    userUPP );
```

userUPP

A ComponentRoutineUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Components.h

DisposeDataHCompletionUPP

Disposes of a DataHCompletionUPP pointer.

```
void DisposeDataHCompletionUPP (  
    DataHCompletionUPP    userUPP );
```

userUPP

A DataHCompletionUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeDoMCActionUPP

Disposes of a DoMCActionUPP pointer.

```
void DisposeDoMCActionUPP (  
    DoMCActionUPP    userUPP );
```

DisposeEndDocumentHandlerUPP

userUPP

A DoMCActionUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

DisposeEndDocumentHandlerUPP

Disposes of an EndDocumentHandlerUPP pointer.

```
void DisposeEndDocumentHandlerUPP (  
    EndDocumentHandlerUPP    userUPP ); QuickTimeComponents.h
```

userUPP

An EndDocumentHandlerUPP pointer.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

DisposeEndElementHandlerUPP

Disposes of an EndElementHandlerUPP pointer.

```
void DisposeEndElementHandlerUPP (  
    EndElementHandlerUPP    userUPP );
```

userUPP

An EndElementHandlerUPP pointer.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

DisposeFilePlayCompletionUPP

Deprecated.

```
void DisposeFilePlayCompletionUPP (
    FilePlayCompletionUPP    userUPP );
```

Version Notes

Introduced in QuickTime 4.1. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: Sound.h

DisposeGetMissingComponentResourceUPP

Disposes of a GetMissingComponentResourceUPP pointer.

```
void DisposeGetMissingComponentResourceUPP (
    GetMissingComponentResourceUPP    userUPP );
```

userUPP

A GetMissingComponentResourceUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Components.h

DisposeGetMovieUPP

Disposes of a GetMovieUPP pointer.

```
void DisposeGetMovieUPP (
    GetMovieUPP    userUPP );
```

userUPP

A GetMovieUPP pointer. See “Universal Procedure Pointers” (IV–2641).

DisposeGWorld

function result You can access this function's error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

DisposeGWorld

Disposes of an offscreen graphics world.

```
void DisposeGWorld (
    GWorldPtr    offscreenGWorld );
```

`offscreenGWorld`

A pointer to an offscreen graphics world.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

When you dispose of a graphics world using this function, the graphics world's pixel map is not disposed, because QuickTime does not know how its pixels were originally allocated. It is the caller's responsibility to dispose of the pixel map.

Version Notes

Part of the Mac OS.

Programming Info

C interface file: `QDOffscreen.h`

DisposeICMAAlignmentUPP

Disposes of an `ICMAAlignmentUPP` pointer.

```
void DisposeICMAAlignmentUPP (
    ICMAAlignmentUPP    userUPP );
```

`userUPP`

An `ICMAAlignmentUPP` pointer. See “Universal Procedure Pointers” (IV-2641).

function result You can access this function's error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `ImageCompression.h`

DisposeICMCompletionUPP

Disposes of an `ICMCompletionUPP` pointer.

```
void DisposeICMCompletionUPP (  
    ICMCompletionUPP    userUPP );
```

`userUPP`

An `ICMCompletionUPP` pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `ImageCompression.h`

DisposeICMConvertDataFormatUPP

Disposes of an `ICMConvertDataFormatUPP` pointer.

```
void DisposeICMConvertDataFormatUPP (  
    ICMConvertDataFormatUPP    userUPP );
```

`userUPP`

An `ICMConvertDataFormatUPP` pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `ImageCompression.h`

DisposeICMCursorShieldedUPP

DisposeICMCursorShieldedUPP

Disposes of an ICMCursorShieldedUPP pointer.

```
void DisposeICMCursorShieldedUPP (  
    ICMCursorShieldedUPP    userUPP );
```

userUPP

An ICMCursorShieldedUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: ImageCompression.h

DisposeICMDataUPP

Disposes of an ICMDataUPP pointer.

```
void DisposeICMDataUPP (  
    ICMDataUPP    userUPP );
```

userUPP

An ICMDataUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: ImageCompression.h

DisposeICMFlushUPP

Disposes of an ICMFlushUPP pointer.

```
void DisposeICMFlushUPP (  
    ICMFlushUPP    userUPP );
```

```
ICMFlushUPP    userUPP );
```

userUPP

An ICMFlushUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: ImageCompression.h

DisposeICMMemoryDisposedUPP

Disposes of an ICMemoryDisposedUPP pointer.

```
void DisposeICMMemoryDisposedUPP (  
    ICMemoryDisposedUPP    userUPP );
```

userUPP

An ICMemoryDisposedUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: ImageCompression.h

DisposeICMProgressUPP

Disposes of an ICMProgressUPP pointer.

```
void DisposeICMProgressUPP (  
    ICMProgressUPP    userUPP );
```

userUPP

An ICMProgressUPP pointer. See “Universal Procedure Pointers” (IV–2641).

DisposeImageCodecDrawBandCompleteUPP

function result You can access this function's error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `ImageCompression.h`

DisposeImageCodecDrawBandCompleteUPP

Disposes of an `ImageCodecDrawBandCompleteUPP` pointer.

```
void DisposeImageCodecDrawBandCompleteUPP (  
    ImageCodecDrawBandCompleteUPP    userUPP );
```

`userUPP`

An `ImageCodecDrawBandCompleteUPP` pointer.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCodec.h`

DisposeImageCodecMPDrawBandUPP

Disposes of an `ImageCodecMPDrawBandUPP` pointer.

```
void DisposeImageCodecMPDrawBandUPP (  
    ImageCodecMPDrawBandUPP    userUPP );
```

`userUPP`

An `ImageCodecMPDrawBandUPP` pointer. See “Universal Procedure Pointers” (IV-2641).

function result You can access this function's error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `ImageCodec.h`

DisposeImageCodecTimeTriggerUPP

Disposes of an ImageCodecTimeTriggerUPP pointer.

```
void DisposeImageCodecTimeTriggerUPP (
    ImageCodecTimeTriggerUPP    userUPP );
```

userUPP

An ImageCodecTimeTriggerUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: ImageCodec.h

DisposeMatte

Disposes of a matte obtained from the GetTrackMatte (I–534) function.

```
void DisposeMatte (
    PixMapHandle    theMatte );
```

theMatte

Handle to the matte to be disposed. Your application obtains this handle from GetTrackMatte (I–534).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

DisposeMCActionFilterUPP

DisposeMCActionFilterUPP

Disposes of a MCActionFilterUPP pointer.

```
void DisposeMCActionFilterUPP (  
    MCActionFilterUPP    userUPP );
```

userUPP

A MCActionFilterUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

DisposeMCActionFilterWithRefConUPP

Disposes of a MCActionFilterWithRefConUPP pointer.

```
void DisposeMCActionFilterWithRefConUPP (  
    MCActionFilterWithRefConUPP    userUPP );
```

userUPP

A MCActionFilterWithRefConUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

DisposeMovie

Frees any memory being used by a movie, including the memory used by the movie’s tracks and media structures.

```
void DisposeMovie (
    Movie    theMovie );
```

theMovie

Identifies the movie to be freed. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), or `NewMovieFromHandle` (II–1084).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Discussion

Your application should call this function when it is done working with a movie, as shown in the following example:

```
// DisposeMovie coding example
// See “Discovering QuickTime,” page 85
void CreateMyCoolMovie (void)
{
    StandardFileReply    sfr;
    Movie                movie = NIL;
    FSSpec                fss;
    short                nFileRefNum = 0;
    short                nResID = movieInDataForkResID;

    StandardPutFile("\pEnter movie file name:", "\puntitled.mov", &sfr);
    if (!sfr.sfGood)
        return;

    CreateMovieFile(&sfr.sfFile,
                   FOUR_CHAR_CODE('TVOD'),
                   smCurrentScript,
                   createMovieFileDeleteCurFile |
createMovieFileDontCreateResFile,
                   &nFileRefNum,
                   &movie);

    CreateMyVideoTrack(movie);        // See “Creating a Track,” below

    AddMovieResource(movie, nFileRefNum, &nResID, NIL);
    if (nFileRefNum != 0)
        CloseMovieFile(nFileRefNum);
    DisposeMovie(movie);
```

DisposeMovieController

```
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Functions” (V-2713)

DisposeMovieController

Disposes of a movie controller.

```
void DisposeMovieController (
    ComponentInstance mc );
```

`mc`

The movie controller for the operation. You obtain this identifier from the Component Manager’s `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131) function, or from the `NewMovieController` (II-1071) function.

function result You can access this function’s error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Discussion

This function is implemented by the Movie Toolbox, not by movie controller components. If you are creating your own movie controller component, you do not have to support this function. The following code snippet illustrates its use:

```
// DisposeMovieController coding example
// See “Discovering QuickTime,” page 221
// Resource identifiers
#define IDM_OPEN 101

char          szMovieFile[MAX_PATH];           // Name of movie file
Movie         movie;                           // Movie object
MovieController mc;                            // Movie controller

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    .
    .
    .
```

```

InitializeQTML(0);                // Initialize QuickTime
EnterMovies();                    // Initialize Toolbox
.
// Main message loop
.
ExitMovies();                     // Terminate Toolbox
TerminateQTML();                 // Terminate QuickTime
} // end WinMain
//
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    MSG                msg;
    EventRecord        er;

    . . .                    // Fill in contents of MSG structure

    WinEventToMacEvent(&msg, &er);                // Convert message to a
QT event
    MCIsPlayerEvent(mc, (const EventRecord *)&er); // Pass event to movie
controller

    switch (iMsg) {
        case WM_CREATE:
            CreatePortAssociation(hwnd, NIL, 0L); // Register window with QT
            break;
        case WM_COMMAND:
            switch (LOWORD(wParam)) {
                case IDM_OPEN:
                    MyCloseMovie();                // Close previous movie, if any

                    if (MyGetFile(szMovieFile))    // Get file name from user
                        MyOpenMovie(hwnd, szMovieFile); // Open the movie
                    break;
                . . .
            default:
                return DefWindowProc(hwnd, iMsg, wParam, lParam);
            } // end switch (LOWORD(wParam))
            break;
        case WM_CLOSE:
            DestroyPortAssociation(GetNativeWindowPort(hwnd)); // Unregister
window

```

DisposeMovieController

```
        break;
    . . .
    default:
        return DefWindowProc(hwnd, iMsg, wParam, lParam);

} // end switch (iMsg)

return 0;
} // end WndProc
//
BOOL MyGetFile (char *lpszMovieFile)
{
    OPENFILENAME        ofn;

    // Fill in contents of OPENFILENAME structure
    .
    .

    if (GetOpenFileName(&ofn))                // Let user select file
        return TRUE;
    else
        return FALSE;
} // end MyGetFile
//
void MyOpenMovie (HWND hwnd, char szFileName[255])
{
    short    nFileRefNum = 0;
    FSSpec   fss;

    SetGWorld((CGrafPtr)GetNativeWindowPort(hwnd), NIL); // Set graphics
port
    NativePathNameToFSSpec(szFileName, &fss, 0); // Convert pathname and
make FSSpec
    OpenMovieFile(&fss, &nFileRefNum, fsRdPerm); // Open movie file
    NewMovieFromFile(&movie, nFileRefNum, NIL, // Get movie from file
        NIL, newMovieActive, NIL);
    CloseMovieFile(nFileRefNum); // Close movie file

    mc = NewMovieController(movie, ...); // Make movie controller
    .
    .
}
```

```
} // end MyOpenMovie
//
void MyCloseMovie (void)
{
    if (mc)                                // Destroy movie controller, if any
        DisposeMovieController(mc);

    if (movie)                             // Destroy movie object, if any
        DisposeMovie(movie);
} // end MyCloseMovie
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Associating Movies With Controllers” (V–2774)

DisposeMovieDrawingCompleteUPP

Disposes of a `MovieDrawingCompleteUPP` pointer.

```
void DisposeMovieDrawingCompleteUPP (
    MovieDrawingCompleteUPP    userUPP );
```

`userUPP`

A `MovieDrawingCompleteUPP` pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

DisposeMovieEditState

Disposes of an edit state.

DisposeMovieExecuteWiredActionsUPP

```
OSErr DisposeMovieEditState (
    MovieEditState    state );
```

state

The edit state for this operation. Your application obtains this edit state identifier when you create the edit state by calling `NewMovieEditState` (II–1073).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Special Considerations

You must dispose of a movie’s edit states before you dispose of the movie itself.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Undo for Movies” (V–2748)

DisposeMovieExecuteWiredActionsUPP

Disposes of a `MovieExecuteWiredActionsUPP` pointer.

```
void DisposeMovieExecuteWiredActionsUPP (
    MovieExecuteWiredActionsUPP    userUPP );
```

userUPP

A `MovieExecuteWiredActionsUPP` pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

DisposeMovieExportGetDataUPP

Disposes of a `MovieExportGetDataUPP` pointer.


```
void DisposeMovieExportGetDataUPP (  
    MovieExportGetDataUPP    userUPP );
```

userUPP

A MovieExportGetDataUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeMovieExportGetPropertyUPP

Disposes of a MovieExportGetPropertyUPP pointer.

```
void DisposeMovieExportGetPropertyUPP (  
    MovieExportGetPropertyUPP    userUPP );
```

userUPP

A MovieExportGetPropertyUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeMoviePrePrerollCompleteUPP

Disposes of a MoviePrePrerollCompleteUPP pointer.

```
void DisposeMoviePrePrerollCompleteUPP (  
    MoviePrePrerollCompleteUPP    userUPP );
```

DisposeMoviePreviewCallOutUPP

userUPP

A MoviePrePrerollCompleteUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

DisposeMoviePreviewCallOutUPP

Disposes of a MoviePreviewCallOutUPP pointer.

```
void DisposeMoviePreviewCallOutUPP (  
    MoviePreviewCallOutUPP    userUPP );
```

userUPP

A MoviePreviewCallOutUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

DisposeMovieProgressUPP

Disposes of a MovieProgressUPP pointer.

```
void DisposeMovieProgressUPP (  
    MovieProgressUPP    userUPP );
```

userUPP

A MovieProgressUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError

(I-492) and GetMoviesStickyError (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

DisposeMovieRgnCoverUPP

Disposes of a `MovieRgnCoverUPP` pointer.

```
void DisposeMovieRgnCoverUPP (  
    MovieRgnCoverUPP    userUPP );
```

`userUPP`

A `MovieRgnCoverUPP` pointer. See “Universal Procedure Pointers” (IV-2641).

function result You can access this function’s error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

DisposeMoviesErrorUPP

Disposes of a `MoviesErrorUPP` pointer.

```
void DisposeMoviesErrorUPP (  
    MoviesErrorUPP    userUPP );
```

`userUPP`

A `MoviesErrorUPP` pointer. See “Universal Procedure Pointers” (IV-2641).

function result You can access this function’s error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

DisposeMovieTrack

DisposeMovieTrack

Removes a track from a movie.

```
void DisposeMovieTrack (  
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II-1092) and `GetMovieTrack` (I-497).

function result You can access this function's error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Discussion

The following code snippet illustrates the use of `DisposeMovieTrack`:

```
// DisposeMovieTrack coding example  
// See "Discovering QuickTime," page 363  
Movie          movie1;  
TimeValue      101dDuration;  
Movie          movie2;  
long           1Index, 1OrigTrackCount, 1ReferenceIndex;  
Track          track, trackSprite;  
  
// get the first track in original movie and position at the start  
trackSprite = GetMovieIndTrack(movie1, 1);  
SetMovieSelection(movie1, 0, 0);  
  
// remove all tracks except video in modifier movie  
for (1Index = 1; 1Index <= GetMovieTrackCount(movie2); 1Index++) {  
    Track      track = GetMovieIndTrack(movie2, 1Index);  
    OSType     dwType;  
  
    GetMediaHandlerDescription(GetTrackMedia(track),  
                              &dwType, NIL, NIL);  
    if (dwType != VideoMediaType) {  
        DisposeMovieTrack(track);  
        1Index--;  
    }  
}  
  
// add the modifier track to original movie
```

```
10ldDuration = GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);

// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, 10ldDuration,
                    GetMovieDuration(movie1) - 10ldDuration);

// associate the modifier track with the original sprite track
track = GetMovieIndTrack(movie1, 10origTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                  &lReferenceIndex);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating Tracks and Media Structures” (V–2726)

Related Java Methods

`quicktime.std.movies.Movie.removeTrack()`

DisposeMusicMIDIReadHookUPP

Disposes of a `MusicMIDIReadHookUPP` pointer.

```
void DisposeMusicMIDIReadHookUPP (
    MusicMIDIReadHookUPP    userUPP );
```

userUPP

A `MusicMIDIReadHookUPP` pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `QuickTimeMusic.h`

DisposeMusicMIDISendUPP

DisposeMusicMIDISendUPP

Disposes of a MusicMIDISendUPP pointer.

```
void DisposeMusicMIDISendUPP (  
    MusicMIDISendUPP    userUPP );
```

userUPP

A MusicMIDISendUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeMusic.h

DisposeMusicOfflineDataUPP

Disposes of a MusicOfflineDataUPP pointer.

```
void DisposeMusicOfflineDataUPP (  
    MusicOfflineDataUPP    userUPP );
```

userUPP

A MusicOfflineDataUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeMusic.h

DisposePrePrerollCompleteUPP

Disposes of a PrePrerollCompleteUPP pointer.

```
void DisposePrePrerollCompleteUPP (  
    PrePrerollCompleteUPP    userUPP );
```

userUPP

A PrePrerollCompleteUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: MediaHandlers.h

DisposePreprocessInstructionHandlerUPP

Disposes of a PreprocessInstructionHandlerUPP pointer.

```
void DisposePreprocessInstructionHandlerUPP (  
    PreprocessInstructionHandlerUPP    userUPP );
```

userUPP

A PreprocessInstructionHandlerUPP pointer.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

DisposeQDPixUPP

Disposes of a QDPixUPP pointer.

```
void DisposeQDPixUPP (  
    QDPixUPP    userUPP );
```

userUPP

A QDPixUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

DisposeQTBandwidthNotificationUPP

Programming Info

C interface file: ImageCompression.h

DisposeQTBandwidthNotificationUPP

Disposes of a QTBandwidthNotificationUPP pointer.

```
void DisposeQTBandwidthNotificationUPP (  
    QTBandwidthNotificationUPP    userUPP );
```

userUPP

A QTBandwidthNotificationUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

DisposeQTCallbackUPP

Disposes of a QTCallbackUPP pointer.

```
void DisposeQTCallbackUPP (  
    QTCallbackUPP    userUPP );
```

userUPP

A QTCallbackUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

DisposeQTSTNotificationUPP

Disposes of a QTSTNotificationUPP pointer.

```
void DisposeQTSTNotificationUPP (  
    QTSTNotificationUPP    userUPP );
```

userUPP

A QTSTNotificationUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h

DisposeQTSTModalFilterUPP

Disposes of a QTSTModalFilterUPP pointer.

```
void DisposeQTSTModalFilterUPP (  
    QTSTModalFilterUPP    userUPP );
```

userUPP

A QTSTModalFilterUPP pointer.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

DisposeQTSPanelFilterUPP

Disposes of a QTSPanelFilterUPP pointer.

```
void DisposeQTSPanelFilterUPP (  
    QTSPanelFilterUPP    userUPP );
```

DisposeQTSyncTaskUPP

userUPP

A QTSPanelFilterUPP pointer.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

DisposeQTSyncTaskUPP

Disposes of a QTSyncTaskUPP pointer.

```
void DisposeQTSyncTaskUPP (
    QTSyncTaskUPP    userUPP );
```

userUPP

A QTSyncTaskUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

DisposeQTVRBackBufferImagingUPP

Disposes of a QTVRBackBufferImagingUPP pointer.

```
void DisposeQTVRBackBufferImagingUPP (
    QTVRBackBufferImagingUPP    userUPP );
```

userUPP

A QTVRBackBufferImagingUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeVR.h

DisposeQTVREnteringNodeUPP

Disposes of a QTVREnteringNodeUPP pointer.

```
void DisposeQTVREnteringNodeUPP (  
    QTVREnteringNodeUPP    userUPP );
```

userUPP

A QTVREnteringNodeUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeVR.h

DisposeQTVRImagingCompleteUPP

Disposes of a QTVRImagingCompleteUPP pointer.

```
void DisposeQTVRImagingCompleteUPP (  
    QTVRImagingCompleteUPP    userUPP );
```

userUPP

A QTVRImagingCompleteUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeVR.h

DisposeQTVRInterceptUPP

DisposeQTVRInterceptUPP

Disposes of a QTVRInterceptUPP pointer.

```
void DisposeQTVRInterceptUPP (  
    QTVRInterceptUPP    userUPP );
```

userUPP

A QTVRInterceptUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeVR.h

DisposeQTVRLeavingNodeUPP

Disposes of a QTVRLeavingNodeUPP pointer.

```
void DisposeQTVRLeavingNodeUPP (  
    QTVRLeavingNodeUPP    userUPP );
```

userUPP

A QTVRLeavingNodeUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeVR.h

DisposeQTVRMouseOverHotSpotUPP

Disposes of a QTVRMouseOverHotSpotUPP pointer.

```
void DisposeQTVRMouseOverHotSpotUPP (  
    QTVRMouseOverHotSpotUPP    userUPP );
```

userUPP

A QTVRMouseOverHotSpotUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeVR.h

DisposeRTPMPDataReleaseUPP

Disposes of an RTPMPDataReleaseUPP pointer.

```
void DisposeRTPMPDataReleaseUPP (  
    RTPMPDataReleaseUPP    userUPP );
```

userUPP

An RTPMPDataReleaseUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QTStreamingComponents.h

DisposeRTPPBCallbackUPP

Disposes of an RTPPBCallbackUPP pointer.

```
void DisposeRTPPBCallbackUPP (  
    RTPPBCallbackUPP    userUPP );
```

userUPP

An RTPPBCallbackUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError

DisposeSCModalFilterUPP

(I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QTStreamingComponents.h

DisposeSCModalFilterUPP

Disposes of an SCModalFilterUPP pointer.

```
void DisposeSCModalFilterUPP (  
    SCModalFilterUPP    userUPP );
```

userUPP

An SCModalFilterUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSCModalHookUPP

Disposes of an SCModalHookUPP pointer.

```
void DisposeSCModalHookUPP (  
    SCModalHookUPP    userUPP );
```

userUPP

An SCModalHookUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSGAddFrameBottleUPP

Disposes of an SGAddFrameBottleUPP pointer.

```
void DisposeSGAddFrameBottleUPP (
    SGAddFrameBottleUPP    userUPP );
```

userUPP

An SGAddFrameBottleUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSGCompressBottleUPP

Disposes of an SGCompressBottleUPP pointer.

```
void DisposeSGCompressBottleUPP (
    SGCompressBottleUPP    userUPP );
```

userUPP

An SGCompressBottleUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSGCompressCompleteBottleUPP

Disposes of an SGCompressCompleteBottleUPP pointer.

DisposeSGDataUPP

```
void DisposeSGCompressCompleteBottleUPP (  
    SGCompressCompleteBottleUPP    userUPP );
```

userUPP

An SGCompressCompleteBottleUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSGDataUPP

Disposes of an SGDataUPP pointer.

```
void DisposeSGDataUPP (  
    SGDataUPP    userUPP );
```

userUPP

An SGDataUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSGDisplayBottleUPP

Disposes of an SGDisplayBottleUPP pointer.

```
void DisposeSGDisplayBottleUPP (  
    SGDisplayBottleUPP    userUPP );
```

userUPP

An SGDisplayBottleUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function's error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSGDisplayCompressBottleUPP

Disposes of an SGDisplayCompressBottleUPP pointer.

```
void DisposeSGDisplayCompressBottleUPP (  
    SGDisplayCompressBottleUPP    userUPP );
```

userUPP

An SGDisplayCompressBottleUPP pointer. See “Universal Procedure Pointers” (IV-2641).

function result You can access this function's error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSGGrabBottleUPP

Disposes of an SGGrabBottleUPP pointer.

```
void DisposeSGGrabBottleUPP (  
    SGGrabBottleUPP    userUPP );
```

userUPP

An SGGrabBottleUPP pointer. See “Universal Procedure Pointers” (IV-2641).

function result You can access this function's error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494).

Version Notes

Introduced in QuickTime 4.1.

DisposeSGGrabCompleteBottleUPP

Programming Info

C interface file: QuickTimeComponents.h

DisposeSGGrabCompleteBottleUPP

Disposes of an SGGrabCompleteBottleUPP pointer.

```
void DisposeSGGrabCompleteBottleUPP (  
    SGGrabCompleteBottleUPP    userUPP );
```

userUPP

An SGGrabCompleteBottleUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSGGrabCompressCompleteBottleUPP

Disposes of an SGGrabCompressCompleteBottleUPP pointer.

```
void DisposeSGGrabCompressCompleteBottleUPP (  
    SGGrabCompressCompleteBottleUPP    userUPP );
```

userUPP

An SGGrabCompressCompleteBottleUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSGModalFilterUPP

Disposes of an SGModalFilterUPP pointer.

```
void DisposeSGModalFilterUPP (
    SGModalFilterUPP    userUPP );
```

userUPP

An SGModalFilterUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSGTransferFrameBottleUPP

Disposes of an SGTransferFrameBottleUPP pointer.

```
void DisposeSGTransferFrameBottleUPP (
    SGTransferFrameBottleUPP    userUPP );
```

userUPP

An SGTransferFrameBottleUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeComponents.h

DisposeSICompletionUPP

Disposes of an SICompletionUPP pointer.

```
void DisposeSICompletionUPP (
```

DisposeSIInterruptUPP

```
SICompletionUPP    userUPP );
```

userUPP

An SICompletionUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Sound.h

DisposeSIInterruptUPP

Disposes of an SIInterruptUPP pointer.

```
void DisposeSIInterruptUPP (  
    SIInterruptUPP    userUPP );
```

userUPP

An SIInterruptUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Sound.h

DisposeSndCallbackUPP

Disposes of a SndCallbackUPP pointer.

```
void DisposeSndCallbackUPP (  
    SndCallbackUPP    userUPP );
```

userUPP

A SndCallbackUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError

(I-492) and GetMoviesStickyError (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Sound.h

DisposeSndDoubleBackUPP

Deprecated.

```
void DisposeSndDoubleBackUPP (  
    SndDoubleBackUPP    userUPP );
```

Version Notes

Introduced in QuickTime 4.1. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: Sound.h

DisposeSoundConverterFillBufferDataUPP

Disposes of a SoundConverterFillBufferDataUPP pointer.

```
void DisposeSoundConverterFillBufferDataUPP (  
    SoundConverterFillBufferDataUPP    userUPP );
```

userUPP

A SoundConverterFillBufferDataUPP pointer. See “Universal Procedure Pointers” (IV-2641).

function result You can access this function’s error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Sound.h

DisposeSoundParamUPP

DisposeSoundParamUPP

Disposes of a SoundParamUPP pointer.

```
void DisposeSoundParamUPP (  
    SoundParamUPP    userUPP );
```

userUPP

A SoundParamUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Sound.h

DisposeSprite

Disposes of a **sprite**.

```
void DisposeSprite (  
    Sprite    theSprite );
```

theSprite

The **sprite** to be disposed of.

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Discussion

You call this function to dispose of a **sprite** created by NewSprite (II–1113). The image description handle and image data pointer associated with the sprite are not disposed of by this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Creating and Manipulating Sprites” (V–2901)

Related Java Methods

quicktime.std.anim.Sprite.remove()

DisposeSpriteWorld

Disposes of a **sprite** world.

```
void DisposeSpriteWorld (
    SpriteWorld    theSpriteWorld );
```

theSpriteWorld

The **sprite** world to dispose of. It is safe to pass NIL to this function.

function result You can access this function's error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494).

Discussion

You call this function to dispose of a **sprite** world created by NewSpriteWorld (II-1114). This function also disposes of all of the sprites associated with the sprite world. This function does not dispose of the graphics worlds associated with the sprite world. Here is an example of using it:

```
// DisposeSpriteWorld coding example
// See "Discovering QuickTime," page 347
#define kNumSprites          4
#define kNumSpaceShipImages  24

SpriteWorld    gSpriteWorld = NIL;
Sprite          gSprites[kNumSprites];
Handle          gCompressedPictures[kNumSpaceShipImages];
ImageDescriptionHandle gImageDescriptions[kNumSpaceShipImages];

void MyDisposeEverything (void)
{
    short        nIndex;

    // dispose of the sprite world and associated graphics world
    if (gSpriteWorld)
        DisposeSpriteWorld(gSpriteWorld);

    // dispose of each sprite's image data
    for (nIndex = 0; nIndex < kNumSprites; nIndex++) {
        if (gCompressedPictures[nIndex])
            DisposeHandle(gCompressedPictures[nIndex]);
        if (gImageDescriptions[nIndex])
            DisposeHandle((Handle)gImageDescriptions[nIndex]);
    }
}
```

DisposeStartDocumentHandlerUPP

```
    }  
    DisposeGWorld(spritePlane);  
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working with Sprite Worlds” (V–2900)

DisposeStartDocumentHandlerUPP

Disposes of a `StartDocumentHandlerUPP` pointer.

```
void DisposeStartDocumentHandlerUPP (  
    StartDocumentHandlerUPP    userUPP );
```

`userUPP`

A `StartDocumentHandlerUPP` pointer.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

DisposeStartElementHandlerUPP

Disposes of a `StartElementHandlerUPP` pointer.

```
void DisposeStartElementHandlerUPP (  
    StartElementHandlerUPP    userUPP );
```

`userUPP`

A `StartElementHandlerUPP` pointer.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

DisposeStdPixUPP

Disposes of a StdPixUPP pointer.

```
void DisposeStdPixUPP (
    StdPixUPP    userUPP );
```

userUPP

A StdPixUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: ImageCompression.h

DisposeTextMediaUPP

Disposes of a TextMediaUPP pointer.

```
void DisposeTextMediaUPP (
    TextMediaUPP    userUPP );
```

userUPP

A TextMediaUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

DisposeTimeBase

Disposes of a time base once you are finished with it.

```
void DisposeTimeBase (
    TimeBase    tb );
```

DisposeTrackEditState

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II–1119).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Disposing of Time Bases” (V–2759)

DisposeTrackEditState

Disposes of a movie’s track edit state.

```
OSErr DisposeTrackEditState (  
    TrackEditState  state );
```

state

The edit state for this operation. Your application obtains this edit state identifier when you create the edit state by calling the `NewTrackEditState` (II–1119) function.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Your application must dispose of any edit states you create. You create an edit state by calling `NewTrackEditState` (II–1119).

Special Considerations

You must dispose of a movie’s track edit states before you dispose of the track or the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Undo for Tracks” (V–2751)

DisposeTrackMedia

Removes a media from a track.

```
void DisposeTrackMedia (
    Media    theMedia );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535) and.

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Discussion

This function does not remove the track from its movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating Tracks and Media Structures” (V–2726)

Related Java Methods

`quicktime.std.movies.Track.removeMedia()`

DisposeTrackTransferUPP

Disposes of a `TrackTransferUPP` pointer.

```
void DisposeTrackTransferUPP (
    TrackTransferUPP    userUPP );
```

userUPP

A `TrackTransferUPP` pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

DisposeTuneCallbackUPP

DisposeTuneCallbackUPP

Disposes of a TuneCallbackUPP pointer.

```
void DisposeTuneCallbackUPP (  
    TuneCallbackUPP    userUPP );
```

userUPP

A TuneCallbackUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeMusic.h

DisposeTunePlayCallbackUPP

Disposes of a TunePlayCallbackUPP pointer.

```
void DisposeTunePlayCallbackUPP (  
    TunePlayCallbackUPP    userUPP );
```

userUPP

A TunePlayCallbackUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeMusic.h

DisposeTweenDataUPP

Disposes of a TweenDataUPP pointer.

```
void DisposeTweenDataUPP (  
    TweenDataUPP    userUPP );
```

userUPP

A TweenUserDataUPP pointer. See “Universal Procedure Pointers” (IV–2641).

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

DisposeUserData

Disposes of a user data structure created by NewUserData (II–1124).

```
OSErr DisposeUserData (
    UserData    theUserData );
```

theUserData

The user data structure that is to be disposed of. It is acceptable but unnecessary to pass NIL in this parameter.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Working With Movie User Data” (V–2743)

DisposeVdigIntUPP

Disposes of a VdigIntUPP pointer.

```
void DisposeVdigIntUPP (
    VdigIntUPP    userUPP );
```

userUPP

A VdigIntUPP pointer. See “Universal Procedure Pointers” (IV–2641).

DragAlignedGrayRgn

function result You can access this function's error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `QuickTimeComponents.h`

DragAlignedGrayRgn

Drags a specified gray region along an optimal alignment grid.

```
long DragAlignedGrayRgn (
    RgnHandle          theRgn,
    Point              startPt,
    Rect               *boundsRect,
    Rect               *slopRect,
    short              axis,
    UniversalProcPtr    actionProc,
    Rect               *alignmentRect,
    ICMAalignmentProcRecordPtr alignmentProc );
```

theRgn

A handle to the specified region for this operation. When the user holds down the mouse button, `DragAlignedGrayRgn` pulls a gray outline of the region around following the movement of the mouse until the mouse button is released.

startPt

The point where the mouse button was originally pressed in the local coordinates of the current graphics port.

boundsRect

A pointer to the boundary rectangle of the current graphics port. The offset point follows the mouse location except that `DragAlignedGrayRgn` never moves the offset point outside this rectangle. This limits the travel of the region's outline, not the movements of the mouse.

slopRect

A pointer to the slop rectangle that completely encloses the boundary rectangle so that the user is allowed some flexibility in moving the mouse.

axis

Allows you to constrain the region's motion to only one axis (see constants below).

actionProc

Points to a function that defines some action to be performed repeatedly as long as the user holds down the mouse button. The function should have no parameters. If the `actionProc` parameter is `NIL`, `DragAlignedGrayRgn` simply retains control until the mouse button is released.

alignmentRect

A pointer to a rectangle within the bounds of the region specified in the parameter `theRgn`. Pass `NIL` to align using the bounds of the parameter `theRgn`.

alignmentProc

A pointer to your alignment behavior function; see `ICMAAlignmentProc` (III–2086). Pass `NIL` to use the standard behavior.

function result

The difference between the point where the mouse button was pressed and the offset point; that is, the point in the region whose horizontal and vertical offsets from the upper-left corner of the region’s enclosing rectangle are the same as the offsets of the starting point when the user pressed the mouse button. The vertical difference between the starting point and the offset point is stored in the high-order word of the return value and the horizontal difference is stored in the low-order word.

axis Constants

0x0000

No constraint.

0x0001

Constraint along the horizontal axis.

0x0002

Constraint along the vertical axis.

Discussion

This function limits the movement of the region defined by `theRgn` according to the constraints set by the `boundsRect` and `slopRect` parameters. While the cursor is inside the `boundsRect` rectangle, the region’s outline follows it normally. If the mouse button is released while the cursor is within this rectangle, the return value reflects the simple distance that the cursor moved in each dimension. When the cursor moves outside the `boundsRect` rectangle, the offset point stops at the edge of the `boundsRect` rectangle. If the mouse button is released while the cursor is outside the `boundsRect` rectangle but inside the `slopRect` rectangle, the return value reflects only the difference between the starting point and the offset point, regardless of how far outside of the `boundsRect` rectangle the cursor may have moved. (Note that part of the region can fall outside the `boundsRect` rectangle, but not the offset point.)

DragAlignedWindow

When the cursor moves outside the `slopRect` rectangle, the region's outline disappears from the screen. `DragAlignedGrayRgn` continues to track the cursor, however, and if the cursor moves back into the `slopRect` rectangle, the outline reappears. If the mouse button is released while the cursor is anywhere inside the `slopRect` rectangle, the window is redrawn in its new location, calculated from the value returned by `DragAlignedGrayRgn`. If the mouse button is released while the cursor is outside the `slopRect` rectangle, both words of the return value are set to 0x8000 and the window does not move from its original location.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: "Aligning Windows" (V-2797)

DragAlignedWindow

Drags the specified window along an optimal alignment grid.

```
void DragAlignedWindow (
    WindowPtr          wp,
    Point              startPt,
    Rect                *boundsRect,
    Rect                *alignmentRect,
    ICMAlignmentProcRecordPtr alignmentProc );
```

`wp`

A window pointer to the window to be dragged.

`startPt`

A point that is equal to the point where the mouse button was pressed (in global coordinates, as stored in the `where` field of the event structure).

`DragAlignedWindow` pulls a gray outline of the window around the screen, following the movements of the mouse until the button is released.

`boundsRect`

Points to the boundary rectangle in global coordinates. If the mouse button is released when the mouse position is outside the limits of the boundary rectangle, `DragAlignedWindow` returns without moving the window or making it the active window. For a document window, the boundary rectangle typically is four pixels in from the menu bar and from the other edges of the screen, to ensure that there won't be less than a four-pixel-square area of the title bar visible on the screen.

`alignmentRect`

Points to a rectangle in window coordinates that allows you to align the window to a rectangle within the window. Set this parameter to `NIL` to align using the bounds of the window.

`alignmentProc`

A pointer to your alignment behavior function; see `ICMAAlignmentProc` (III–2086). Pass `NIL` to use the standard behavior.

Discussion

The following code sample illustrates the use of `DragAlignedWindow`:

```
// DragAlignedWindow coding example
// See "Discovering QuickTime," page 265
Boolean IsQuickTimeInstalled (void)
{
    OSErr      nErr;
    long       lResult;

    nErr = Gestalt(gestaltQuickTime, &lResult);
    return (nErr == noErr);
}

void MyInitialize (void)
{
    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NIL);
    MaxApplZone();

    EnterMovies();
}

WindowPtr MakeMyWindow (void)
{
    WindowPtr  pMacWnd;
    Rect       rectWnd = {0, 0, 120, 160};
    Rect       rectBest;
```

DragAlignedWindow

```
// figure out the best monitor for the window
GetBestDeviceRect(NIL, &rectBest);

// put the window in the top left corner of that monitor
OffsetRect(&rectWnd, rectBest.left + 10, rectBest.top + 50);

// create the window
pMacWnd = NewCWindow(NIL, &rectWnd, "\\pGrabber",
                    TRUE, noGrowDocProc, (WindowPtr)-1,
                    TRUE, 0);

// set the port to the new window
SetPort(pMacWnd);

return pMacWnd;
}

main (void)
{
    WindowPtr      pMacWnd;
    SeqGrabComponent seqGrab;
    SGChannel       sgchanVideo, sgchanSound;
    Boolean         bDone = FALSE;
    OSErr           nErr;

    MyInitialize();
    pMacWnd = MakeMyWindow();
    seqGrab = MakeMySequenceGrabber(pMacWnd);
    if (seqGrab == NIL)
        return;

    MakeMyGrabChannels(seqGrab, &sgchanVideo, &sgchanSound,
                      &pMacWnd->portRect, FALSE);

    nErr = SGStartPreview(seqGrab);

    while (!bDone) {
        ICMAlignmentProcRecord apr;
        short                  nPart;
        WindowPtr              pWhichWnd;
        EventRecord             er;
```

```

GetNextEvent(everyEvent, &er);

switch (er.what) {
    case nullEvent:      // give the sequence grabber time
        nErr = SGIdle(seqGrab);
        if (nErr != noErr)
            bDone = TRUE;
        break;

    case updateEvt:
        if (er.message == (long)pMacWnd) {
            // inform the sequence grabber of the update
            SGUpdate(seqGrab,((WindowPeek)
                               pMacWnd)->updateRgn);
            // and swallow the update event
            BeginUpdate(pMacWnd);
            EndUpdate(pMacWnd);
        }
        break;

    case mouseDown:
        nPart = FindWindow(er.where, &pWhichWnd);
        if (pWhichWnd != pMacWnd)
            break;
        switch (nPart) {
            case inContent:
                // pause until mouse button is released
                SGPause(seqGrab, TRUE);
                while (StillDown())
                    SGPause(seqGrab, FALSE);
                break;
            case inGoAway:
                bDone = TrackGoAway(pMacWnd, er.where);
                break;

            case inDrag:
                // pause when dragging window so video
                // doesn't draw in the wrong place
                SGPause(seqGrab, TRUE);

```

DrawPictureFile

```
                SGGetAlignmentProc(seqGrab, &apr);
                DragAlignedWindow(pMacWnd,
                                er.where,
                                &screenBits.bounds,
                                NIL, &alignProc);
                SGPause(seqGrab, FALSE);
                break;
            }
        break;
    }

    // clean up
    SGStop(seqGrab);
    CloseComponent(seqGrab);
    DisposeWindow(pMacWnd);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Aligning Windows” (V-2797)

DrawPictureFile

Draws an image from a specified **picture file** in the current graphics port.

```
OSErr DrawPictureFile (
    short                refNum,
    const Rect           *frame,
    ICMProgressProcRecordPtr progressProc );
```

refNum

A file reference number for the source PICT file.

frame

A pointer to the rectangle into which the image is to be loaded. The compressor scales the source image to fit into this destination rectangle.

progressProc

Points to an ICMProgressProc (III–2093) callback. During the operation, the draw function may occasionally call a function you provide in order to report its progress; see ICMProgressProcRecord (IV–2284). If you have not provided a **progress function**, set this parameter to NIL. If you pass a value of –1, QuickTime provides a standard progress function.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function draws the picture that it finds in the **picture file** specified by the refNum parameter within the rectangle specified by the frame parameter. The Image Compression Manager performs any spooling that may be necessary when reading the picture file. Specify a clipping region appropriate for your picture before drawing it. If the clipping region is very large (as it is when a graphics port is initialized) and you want to scale the picture, the clipping region can become invalid when DrawPictureFile scales the clipping region, in which case your picture will not be drawn. On the other hand, if the graphics port specifies a small clipping region, part of your drawing may be clipped when DrawPictureFile draws it. Setting a clipping region equal to the port rectangle of the current graphics port always sets a valid clipping region.

Special Considerations

When it scales fonts, DrawPictureFile changes the size of the font instead of scaling the bits. However, the widths used by bitmap fonts are not always linear. For example, the 12-point width isn’t exactly 1/2 of the 24-point width. This can cause lines of text to become slightly longer or shorter as the picture is scaled. The easiest way to avoid such problems is to specify a destination rectangle that is the same size as the bounding rectangle for the picture.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Working With Pictures and PICT Files” (V–2790)

DrawTrimmedPicture

Draws an image that is stored as a picture into the current graphics port and trims that image to fit a specified region.

```
OSErr DrawTrimmedPicture (
    PicHandle          srcPicture,
```

DrawTrimmedPicture

```
const Rect          *frame,  
RgnHandle           trimMask,  
short               doDither,  
ICMProgressProcRecordPtr progressProc );
```

srcPicture

A handle to the source image, stored as a picture.

frame

A pointer to the rectangle into which the decompressed image is to be loaded.

trimMask

A handle to a clipping region in the destination coordinate system. The decompressor applies this mask to the destination image and ignores any image data that fall outside the specified region. Set this parameter to `NIL` if you do not want to clip the source image.

doDither

Indicates whether to **dither** the image. Use this parameter if you want the image to be dithered when it is displayed on a lower-resolution screen (see below).

progressProc

A pointer to an `ICMProgressProc` (III–2093) callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a **progress function**, set this parameter to `NIL`. If you pass a value of `-1`, QuickTime provides a standard progress function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

doDither Constants

defaultDither

The **dithering** in the source file is to be respected.

forceDither

The specified image should be **dithered** whether the source uses dithering or not.

suppressDither

Dithering should not be used in any case. The ability to suppress **dithering** can be useful if, for example, you have a 32-bit, color JPEG image drawn into an 8-bit buffer with a custom color table from the image. In that case, dithering would not be necessary and probably not desirable, particularly if the buffer were to be compressed with the graphics compressor.

Discussion

This function works with compressed image data; the source data stays compressed. The function trims the image to fit the specified clipping region. Note that if you just use a clip while making a picture, the data (though not visible) is still stored in the picture.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Working With Pictures and PICT Files” (V–2790)

Related Java Methods

`quicktime.qd.Pict.drawTrimmed()`

DrawTrimmedPictureFile

Draws an image that is stored as a **picture file** into the current graphics port and trims that image to fit a specified region.

```
OSErr DrawTrimmedPictureFile (
    short          srcRefnum,
    const Rect     *frame,
    RgnHandle      trimMask,
    short          doDither,
    ICMProgressProcRecordPtr progressProc );
```

`srcRefnum`

A file reference number for the source PICT file.

`frame`

A pointer to the rectangle into which the decompressed image is to be loaded.

`trimMask`

A handle to a clipping region in the destination coordinate system. The decompressor applies this mask to the destination image and ignores any image data that fall outside the specified region. Set this parameter to `NIL` if you do not want to clip the source image. In this case, this function acts like `DrawPictureFile` (I–310).

`doDither`

Indicates whether to **dither** the image. Use this parameter if you want the image to be dithered when it is displayed on a lower-resolution screen (see below).

EndFullScreen

`progressProc`

A pointer to an `ICMProgressProc` (III–2093) callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a **progress function**, set this parameter to `NIL`. If you pass a value of `-1`, QuickTime provides a standard progress function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

doDither Constants

`defaultDither`

The **dithering** in the source file is to be respected.

`forceDither`

The specified image should be **dithered** whether the source uses dithering or not.

`suppressDither`

Dithering should not be used in any case. The ability to suppress **dithering** can be useful if, for example, you have a 32-bit, color JPEG image drawn into an 8-bit buffer with a custom color table from the image. In that case, dithering would not be necessary and probably not desirable, particularly if the buffer were to be compressed with the graphics compressor.

Discussion

You can use this function to save part of a picture, since the image data that is not within the trim region is ignored and is not included in the destination **picture file**. All the remaining objects in the resulting object are clipped.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pictures and PICT Files” (V–2790)

E

EndFullScreen

Ends full-screen mode for a graphics device.


```
OSErr EndFullScreen (
    Ptr      fullState,
    long     flags );
```

fullState

The pointer to private state information returned by a previous call to `BeginFullScreen` (I-52).

flags

Reserved. Set this parameter to NIL.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

This function restores the graphics device and other settings to the state specified by the private state information pointed to by the `fullState` parameter. The resulting state is that that was in effect prior to the immediately previous call to `BeginFullScreen` (I-52). The following code illustrates its use:

```
OSErr QTFullScreen_RestoreScreen (void)
{
    OSErr      myErr = noErr;

    #if TARGET_OS_WIN32
        DestroyPortAssociation((CGrafPtr)gFullScreenWindow);
    #endif

    DisposeMovieController(gMC);
    myErr = EndFullScreen(gRestoreState, 0L);

    return(myErr);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Using the Full Screen” (V-2717)

Related Java Methods

`quicktime.std.movies.FullScreen.end()`

Endian16_Swap

Endian16_Swap

Changes the endian format of an unsigned 16-bit integer.

```
UInt16 Endian16_Swap (  
    UInt16    value );
```

value

An unsigned 16-bit integer input.

function result The unsigned 16-bit integer result.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: Endian.h

Endian32_Swap

Changes the endian format of an unsigned 32-bit integer.

```
UInt32 Endian32_Swap (  
    UInt32    value );
```

value

An unsigned 32-bit integer input.

function result The unsigned 32-bit integer result.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: Endian.h

Endian64_Swap

Changes the endian format of an unsigned 64-bit integer.

```
UInt64 Endian64_Swap (  
    UInt64    value );
```

value

An unsigned 64-bit integer input.

function result The unsigned 64-bit integer result.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `Endian.h`

EndianS16_BtoL

Converts a signed 16-bit **big-endian** value to the equivalent **little-endian** value.

```
SInt16 EndianS16_BtoL (
    SInt16    value );
```

value

A signed 16-bit **big-endian** value.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Endian.h`

EndianS16_BtoN

Converts a signed 16-bit **big-endian** value to the equivalent value in the computer's native format.

```
SInt16 EndianS16_BtoN (
    SInt16    value );
```

value

A signed 16-bit **big-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

EndianS16_LtoB

Programming Info

C interface file: `Endian.h`

EndianS16_LtoB

Converts a signed 16-bit **little-endian** value to the equivalent **big-endian** value.

```
SInt16 EndianS16_LtoB (  
    SInt16    value );
```

value

A signed 16-bit **little-endian** value.

function result The equivalent **big-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Endian.h`

EndianS16_LtoN

Converts a signed 16-bit **little-endian** value to the equivalent value in the computer's native format.

```
SInt16 EndianS16_LtoN (  
    SInt16    value );
```

value

A signed 16-bit **little-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Endian.h`

EndianS16_NtoB

Converts a signed 16-bit value in the computer's native format to the equivalent **big-endian** value.

```
SInt16 EndianS16_NtoB (
    SInt16    value );
```

value

A signed 16-bit value in the computer's native format.

function result The equivalent **big-endian** value.

Discussion

The following code illustrates the use of EndianS16_NtoB:

```
// EndianS16_NtoB coding example
// See "Discovering QuickTime," page 338
void QTMusic_UseMIDIInput (void)
{
    ComponentResult      lErr = noErr;
    MIDIInputExample      mie;
    NoteRequest           noteRequest;
    MusicMIDIReadHookUPP  pfReadHook = NIL;

    // open the note allocator component
    mie.fNoteAllocator = OpenDefaultComponent(kNoteAllocatorComponentType,
0);
    if (mie.fNoteAllocator == NIL)
        goto bail;

    noteRequest.info.flags = 0;
    noteRequest.info.reserved = 0;
    // simultaneous tones
    *(short *)&noteRequest.info.polyphony = EndianS16_NtoB(2);
    *(Fixed *)&noteRequest.info.typicalPolyphony =
EndianU32_NtoB(0x00010000);

    lErr = NASTuffToneDescription(mie.fNoteAllocator,
kInst_AcousticGrandPiano,
                                &noteRequest.tone);

    if (lErr != noErr)
        goto bail;
```

EndianS16_NtoL

```
// allocate a note channel
lErr = NANewNoteChannel(mie.fNoteAllocator, &noteRequest,
&mie.fNoteChannel);
if (lErr != noErr)
    goto bail;

pfReadHook = NewMusicMIDIReadHookProc(QTMusic_ReadHook);
NAUseDefaultMIDIInput(mie.fNoteAllocator,
    NewMusicMIDIReadHookProc(pfReadHook), (long)&mie, 0L);
while (!Button());
NALoseDefaultMIDIInput(mie.fNoteAllocator);

bail:
if (pfReadHook != NIL)
    DisposeRoutineDescriptor(pfReadHook);
if (mie.fNoteAllocator != NIL)
    CloseComponent(mie.fNoteAllocator); // disposes note channel too
}
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianS16_NtoL

Converts a signed 16-bit value in the computer's native format to the equivalent **little-endian** value.

```
SInt16 EndianS16_NtoL (
    SInt16    value );
```

value

A signed 16-bit value in the computer's native format.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianS32_BtoL

Converts a signed 32-bit **big-endian** value to the equivalent **little-endian** value.

```
SInt32 EndianS32_BtoL (
    SInt32    value );
```

value

A signed 32-bit **big-endian** value.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianS32_BtoN

Converts a signed 32-bit **big-endian** value to the equivalent value in the computer's native format.

```
SInt32 EndianS32_BtoN (
    SInt32    value );
```

value

A signed 32-bit **big-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianS32_LtoB

Converts a signed 32-bit **little-endian** value to the equivalent **big-endian** value.

```
SInt32 EndianS32_LtoB (
    SInt32    value );
```

EndianS32_LtoN

value

A signed 32-bit **little-endian** value.

function result The equivalent **big-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianS32_LtoN

Converts a signed 32-bit **little-endian** value to the equivalent value in the computer's native format.

```
SInt32 EndianS32_LtoN (  
    SInt32    value );
```

value

A signed 32-bit **little-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianS32_NtoB

Converts a signed 32-bit value in the computer's native format to the equivalent **big-endian** value.

```
SInt32 EndianS32_NtoB (  
    SInt32    value );
```

value

A signed 32-bit value in the computer's native format.

function result The equivalent **big-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Endian.h`

EndianS32_NtoL

Converts a signed 32-bit value in the computer's native format to the equivalent **little-endian** value.

```
SInt32 EndianS32_NtoL (
    SInt32    value );
```

value

A signed 32-bit value in the computer's native format.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Endian.h`

EndianS64_BtoL

Converts a signed 64-bit **big-endian** value to the equivalent **little-endian** value.

```
SInt64 EndianS64_BtoL (
    SInt64    value );
```

value

A signed 64-bit **big-endian** value.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Endian.h`

EndianS64_BtoN

EndianS64_BtoN

Converts a signed 64-bit **big-endian** value to the equivalent value in the computer's native format.

```
SInt64 EndianS64_BtoN (  
    SInt64    value );
```

value

A signed 64-bit **big-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianS64_LtoB

Converts a signed 64-bit **little-endian** value to the equivalent **big-endian** value.

```
SInt64 EndianS64_LtoB (  
    SInt64    value );
```

value

A signed 64-bit **little-endian** value.

function result The equivalent **big-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianS64_LtoN

Converts a signed 64-bit **little-endian** value to the equivalent value in the computer's native format.

```
SInt64 EndianS64_LtoN (  
    SInt64    value );
```

value

A signed 64-bit **little-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianS64_NtoB

Converts a signed 64-bit value in the computer's native format to the equivalent **big-endian** value.

```
SInt64 EndianS64_NtoB (
    SInt64    value );
```

value

A signed 64-bit value in the computer's native format.

function result The equivalent **big-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianS64_NtoL

Converts a signed 64-bit value in the computer's native format to the equivalent **little-endian** value.

```
SInt64 EndianS64_NtoL (
    SInt64    value );
```

value

A signed 64-bit value in the computer's native format.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

EndianU16_BtoL

Programming Info

C interface file: Endian.h

EndianU16_BtoL

Converts an unsigned 16-bit **big-endian** value to the equivalent **little-endian** value.

```
UInt16 EndianU16_BtoL (  
    UInt16    value );
```

value

An unsigned 16-bit **big-endian** value.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU16_BtoN

Converts an unsigned 16-bit **big-endian** value to the equivalent value in the computer's native format.

```
UInt16 EndianU16_BtoN (  
    UInt16    value );
```

value

An unsigned 16-bit **big-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU16_LtoB

Converts an unsigned 16-bit **little-endian** value to the equivalent **big-endian** value.

```
UInt16 EndianU16_LtoB (
    UInt16    value );
```

value

An unsigned 16-bit **little-endian** value.

function result The equivalent **big-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU16_LtoN

Converts an unsigned 16-bit **little-endian** value to the equivalent value in the computer's native format.

```
UInt16 EndianU16_LtoN (
    UInt16    value );
```

value

An unsigned 16-bit **little-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU16_NtoB

Converts an unsigned 16-bit value in the computer's native format to the equivalent **big-endian** value.

```
UInt16 EndianU16_NtoB (
    UInt16    value );
```

value

An unsigned 16-bit value in the computer's native format.

EndianU16_NtoL

function result The equivalent **big-endian** value.

Discussion

The following code sample illustrates the use of EndianU16_NtoB:

```
// EndianU16_NtoB coding example
// See "Discovering QuickTime," page 360
if (bWithBackgroundPicture) {
    QTAtomContainer      qtacTrackProperties;
    RGBColor             rgbcBackColor;

    rgbcBackColor.red = EndianU16_NtoB(0x8000);
    rgbcBackColor.green = EndianU16_NtoB(0);
    rgbcBackColor.blue = EndianU16_NtoB(0xffff);

    // create a new atom container for sprite track properties
    QTNewAtomContainer(&qtacTrackProperties);

    // add an atom for the background color property
    QTInsertChild(qtacTrackProperties, 0,
        kSpriteTrackPropertyBackgroundColor, 1, 1, sizeof(RGBColor),
        &rgbcBackColor, NIL);

    // set the sprite track's properties
    nErr = SetMediaPropertyAtom(media, qtacTrackProperties);

    QTDisposeAtomContainer(qtacTrackProperties);
}
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU16_NtoL

Converts an unsigned 16-bit value in the computer's native format to the equivalent **little-endian** value.

```
UInt16 EndianU16_NtoL (
    UInt16    value );
```

value

An unsigned 16-bit value in the computer's native format.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU32_BtoL

Converts an unsigned 32-bit **big-endian** value to the equivalent **little-endian** value.

```
UInt32 EndianU32_BtoL (
    UInt32    value );
```

value

An unsigned 32-bit **big-endian** value.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU32_BtoN

Converts an unsigned 32-bit **big-endian** value to the equivalent value in the computer's native format.

```
UInt32 EndianU32_BtoN (
    UInt32    value );
```

value

An unsigned 32-bit **big-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

EndianU32_LtoB

Programming Info

C interface file: Endian.h

EndianU32_LtoB

Converts an unsigned 32-bit **little-endian** value to the equivalent **big-endian** value.

```
UInt32 EndianU32_LtoB (  
    UInt32    value );
```

value

An unsigned 32-bit **little-endian** value.

function result The equivalent **big-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU32_LtoN

Converts an unsigned 32-bit **little-endian** value to the equivalent value in the computer's native format.

```
UInt32 EndianU32_LtoN (  
    UInt32    value );
```

value

An unsigned 32-bit **little-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU32_NtoB

Converts an unsigned 32-bit value in the computer's native format to the equivalent **big-endian** value.

```
UInt32 EndianU32_NtoB (
    UInt32    value );
```

value

An unsigned 32-bit value in the computer's native format.

function result The equivalent **big-endian** value.

Discussion

The following code sample illustrates the use of EndianU32_NtoB:

```
// EndianU32_NtoB coding example
// See "Discovering QuickTime," page 369
{
    QTAtomContainer      qtacTrackProperties;
    RGBColor             rgbcBackColor;

    rgbcBackColor.red     = EndianU16_NtoB(0x8000);
    rgbcBackColor.green   = EndianU16_NtoB(0);
    rgbcBackColor.blue    = EndianU16_NtoB(0xffff);

    QTNewAtomContainer(&qtacTrackProperties);
    QTInsertChild(qtacTrackProperties, 0,
        kSpriteTrackPropertyBackgroundColor, 1, 1, sizeof(RGBColor),
        &rgbcBackColor, NIL);

    // tell the movie controller that this sprite track has actions
    hasActions = TRUE;
    QTInsertChild(qtacTrackProperties, 0,
        kSpriteTrackPropertyHasActions,
        1, 1, sizeof(hasActions), &hasActions, NIL);

    // tell the Sprite Track to generate QTIdleEvents
    idleEventFrequency = EndianU32_NtoB(60);

    QTInsertChild(qtacTrackProperties, 0,
        kSpriteTrackPropertyQTIdleEventsFrequency, 1, 1,
        sizeof(idleEventFrequency), &idleEventFrequency, NIL);
```

EndianU32_NtoL

```
        SetMediaPropertyAtom(media, qtacTrackProperties);

        QTDisposeAtomContainer(qtacTrackProperties);
    }
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU32_NtoL

Converts an unsigned 32-bit value in the computer's native format to the equivalent **little-endian** value.

```
UInt32 EndianU32_NtoL (
    UInt32    value );
```

value

An unsigned 32-bit value in the computer's native format.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU64_BtoL

Converts an unsigned 64-bit **big-endian** value to the equivalent **little-endian** value.

```
UInt64 EndianU64_BtoL (
    UInt64    value );
```

value

An unsigned 64-bit **big-endian** value.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Endian.h`

EndianU64_BtoN

Converts an unsigned 64-bit **big-endian** value to the equivalent value in the computer's native format.

```
UInt64 EndianU64_BtoN (
    UInt64    value );
```

value

An unsigned 64-bit **big-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Endian.h`

EndianU64_LtoB

Converts an unsigned 64-bit **little-endian** value to the equivalent **big-endian** value.

```
UInt64 EndianU64_LtoB (
    UInt64    value );
```

value

An unsigned 64-bit **little-endian** value.

function result The equivalent **big-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Endian.h`

EndianU64_LtoN

EndianU64_LtoN

Converts an unsigned 64-bit **little-endian** value to the equivalent value in the computer's native format.

```
UInt64 EndianU64_LtoN (  
    UInt64    value );
```

value

An unsigned 64-bit **little-endian** value.

function result The equivalent value in the computer's native format.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU64_NtoB

Converts an unsigned 64-bit value in the computer's native format to the equivalent **big-endian** value.

```
UInt64 EndianU64_NtoB (  
    UInt64    value );
```

value

An unsigned 64-bit value in the computer's native format.

function result The equivalent **big-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndianU64_NtoL

Converts an unsigned 64-bit value in the computer's native format to the equivalent **little-endian** value.

```
UInt64 EndianU64_NtoL (  
    UInt64    value );
```

```
UInt64    value );
```

value

An unsigned 64-bit value in the computer's native format.

function result The equivalent **little-endian** value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Endian.h

EndMediaEdits

Ends a media-editing session.

```
OSErr EndMediaEdits (
    Media    theMedia );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as NewTrackMedia (II-1120) and GetTrackMedia (I-535).

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

The following code sample illustrates the use of EndMediaEdits:

```
// EndMediaEdits coding example
// See “Discovering QuickTime,” page 89
void CreateMyVideoTrack (Movie movie)
{
    Track    track;
    Media    media;
    Rect     rect = {0, 0, 100, 320};

    track = NewMovieTrack(movie,
        FixRatio(rect.right, 1),
        FixRatio(rect.bottom, 1),
        kNoVolume);
```

Enqueue

```
media = NewTrackMedia(track,
    VideoMediaType,
    600,                                // video time scale
    NIL, NIL);

BeginMediaEdits(media);
MyAddVideoSamplesToMedia(media, &rect); // assemble data
EndMediaEdits(media);
InsertMediaIntoTrack(track,
    0,                                // track start time
    0,                                // media start time
    GetMediaDuration(media),
    kFix1);                            // normal speed
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Adding Samples to Media Structures” (V–2752)

Related Java Methods

`quicktime.std.movies.media.Media.endEdits()`

Enqueue

Inserts a queue element into a Windows QHdr (IV–2345) structure.

```
void Enqueue (
    QElemPtr    qElement,
    QHdrPtr     qHeader );

qElement
    A pointer to a QElem (IV–2344) structure.
qHeader
    A pointer to a QHdr (IV–2345) structure.
```

Version Notes

Present in QuickTime 3 and later releases.

Programming Info

C interface file: `OSUtils.h`

EnterMovies

Initializes the Movie Toolbox and creates a private storage area for your application.

```
OSErr EnterMovies ( void );
```

function result Be sure to check the value returned by this function before using any other facilities of the Movie Toolbox. See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Before calling any Movie Toolbox functions, you must use `EnterMovies` to initialize the toolbox. Your application may call `EnterMovies` multiple times. The following code sample demonstrates how your application can call the Gestalt Manager to determine whether the Movie Toolbox is installed, using the selector `gestaltQuickTime ('qtim')`, before calling `EnterMovies`:

```
//Using the Gestalt Manager with the Movie Toolbox
```

```
#include <GestaltEqu.h>
```

```
#include <Movies.h>
```

```
Boolean IsQuickTimeInstalled (void)
```

```
{
```

```
    short    error;
```

```
    long     result;
```

```
    error = Gestalt (gestaltQuickTime, &result);
```

```
    return (error == noErr);
```

```
}
```

```
void main (void)
```

```
{
```

```
    Boolean qtInstalled;
```

```
    .
```

```
    .
```

```
    .
```

```
    qtInstalled = IsQuickTimeInstalled ();
```

```
}
```

```
// EnterMovies coding example
```

```
// See “Discovering QuickTime,” page 242
```

EqualMatrix

```
void MyInitMovieToolbox (void)
{
    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NIL);
    EnterMovies();
}

void main (void)
{
    MyInitMovieToolbox();
    CreateMyCoolMovie();
}
```

Special Considerations

You should initialize any other Macintosh managers your application uses before calling `EnterMovies`. You do not need to balance calls to `EnterMovies` with calls to `ExitMovies` (I-340); you need to call `ExitMovies` only if you finish with the Movie Toolbox long before your application is ready to quit.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Initializing the Movie Toolbox” (V-2712)

Related Java Methods

```
quicktime.QTSession.open(),
quicktime.QTSession.enterMovies()
```

EqualMatrix

Compares two matrices and returns a result that indicates whether the matrices are equal.

```
Boolean EqualMatrix (
    const MatrixRecord    *m1,
    const MatrixRecord    *m2 );
```


m1

A pointer to one matrix for the compare operation.

m2

A pointer to the other matrix for the compare operation.

function result TRUE if the matrices are equal, FALSE otherwise.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Managing Matrices” (V-2764)

Related Java Methods

quicktime.std.image.Matrix.equals()

ExecuteCallback

Called by a clock component when it determines that it is time to execute a callback function.

```
void ExecuteCallback (
    QTCallback cb );
```

cb

Specifies the **callback event** for the operation. Your clock component obtains this value from the parameters passed to your ClockCallMeWhen (I-91) function.

function result You can access this function’s error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494).

Discussion

Before calling the application’s function, the ExecuteCallback function cancels the **callback event**. In this manner, the callback event is prevented from executing twice in succession. It is up to the application, or the callback function itself, to reschedule the callback event.

Special Considerations

Your clock component should not release the memory associated with the **callback event** at this time. You should do so only with ClockDisposeCallback (I-94). This is particularly important when a callback function cannot execute at interrupt time, since the Movie Toolbox schedules such functions for invocation at a later time.

ExitMovies

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Toolbox Clock Support Functions” (V–2869)

ExitMovies

Automatically called when an application quits.

```
void ExitMovies ( void );
```

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Discussion

You only need to call this function if you finish with the Movie Toolbox long before your application is ready to quit. When you call `ExitMovies`, the Movie Toolbox releases the private storage (which may be significant) that was allocated when you called `EnterMovies` (I–337). As a general rule, your application seldom uses this function; the following code illustrates an exception:

```
// ExitMovies coding example
// See “Discovering QuickTime,” page 225
int WINAPI WinMain (INSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG          msg;
    HANDLE        hAccelTable;

    if (!hPrevInstance)                // Is there a previous instance?
        if (!(InitApplication(hInstance))) // Register window class
            return FALSE;                // Report failure

    if (InitializeQTML(0) != 0) {        // Initialize QTML
        MessageBox(hwnd, "QuickTime not available", // Notify user
                  "", MB_OK);
        return FALSE;                    // Report failure
    } // end if (InitializeQTML(0) != 0)

    if (EnterMovies() != 0) {            // Initialize QuickTime
```

```

        MessageBox(hwnd, "QuickTime not available",      // Notify user
                    "", MB_OK);

        return FALSE;                                   // Report failure
    } // end if (EnterMovies() != 0)

    if (!(InitInstance(hInstance, nCmdShow)))            // Create main window
        return FALSE;                                   // Report failure

    hAccelTable = LoadAccelerators(hInstance,           // Load accelerator table
                                   MAKEINTRESOURCE(IDR_ACCELSIMPLESDI));

    // Main message loop

    while (GetMessage(&msg, NIL, 0, 0))                 // Retrieve next message
        if (!TranslateAccelerator(msg.hwnd,             // Check for kbd accelerator
                                   hAccelTable, &msg)) {
            TranslateMessage(&msg);                     // Convert virtual key to character
            DispatchMessage(&msg);                     // Send message to window procedure
        } // end if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

    ExitMovies();                                       // Terminate Toolbox
    TerminateQTML();                                   // Terminate QuickTime

    return msg.WParam;
} // end WinMain

```

Special Considerations

Before calling `ExitMovies`, be sure that you have closed your connections to any components that use the Movie Toolbox, such as movie controllers, sequence grabbers, and so on.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Initializing the Movie Toolbox” (V-2712)

Related Java Methods

`quicktime.QTSession.exitMovies()`

Exp1to3

Expands 3-to-1 compressed sound samples to their original size. Use `SoundConverterConvertBuffer` (III-1862) instead, if possible.

```
void Exp1to3 (
    const void      *inBuffer,
    void            *outBuffer,
    unsigned long    cnt,
    StateBlockPtr    inState,
    StateBlockPtr    outState,
    unsigned long    numChannels,
    unsigned long    whichChannel );
```

`inBuffer`

A pointer to a buffer of samples to be compressed.

`outBuffer`

A pointer to a buffer where the samples are to be written.

`cnt`

The number of samples to compress.

`inState`

A pointer to a 128-byte buffer from which the input state of the algorithm is read, or NIL. This buffer should be filled with zeros to initialize the algorithm.

`outState`

A pointer to a 128-byte buffer to which the output state of the algorithm is written, or NIL. This buffer may be the same as that specified by the `inState` parameter.

`numChannels`

The number of channels in the buffer pointed to by the `inBuffer` parameter.

`whichChannel`

The channel to compress when `numChannels` is greater than 1. The value of this parameter must be in the range 1 to `numChannels`.

Discussion

This function expands `cnt` packets of sound stored in the buffer specified by `inBuffer` and places the result in the buffer specified by `outBuffer`, whose size must be at least `cnt` packets times 2 bytes per packet times 3, or `cnt * 6` bytes. If `numChannels` is greater than 1, then the compressed sound must be stored in interleaved format on a packet basis.

If you expand compressed sound data that includes multiple sound channels, you

retain only one channel of sound, which you specify in the `whichChannel` parameter. Thus, if you use the `Exp1to3` procedure to expand three-channel sound, the output buffer will be the same size as the input buffer since only one channel is retained. To retain multiple channels of sound after expansion, you must call the `Exp1to3` procedure for each channel to be expanded and then interleave the expanded sound data on a sample basis.

The `Exp1to3` procedure expands every packet of sampled-sound data to exactly 6 bytes. You can use the `inState` and `outState` parameters to allow the **MACE** compression routines to preserve information about algorithms across calls. Alternatively, you may pass `NIL` state buffers and let the Sound Manager allocate the buffers internally.

Special Considerations

Because the `Exp1to3` procedure might allocate memory, you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 4. Macintosh Note: Not available in **CarbonLib**, but available if InterfaceLib 7.1 or later is installed.

Programming Info

C interface file: `Sound.h`

E

Exp1to6

Expands 6-to-1 compressed sound samples to their original size. Use `SoundConverterConvertBuffer (III-1862)` instead, if possible.

```
void Exp1to6 (
    const void      *inBuffer,
    void            *outBuffer,
    unsigned long    cnt,
    StateBlockPtr    inState,
    StateBlockPtr    outState,
    unsigned long    numChannels,
    unsigned long    whichChannel );
```

`inBuffer`

A pointer to a buffer of samples to be compressed.

`outBuffer`

A pointer to a buffer where the samples are to be written.

FCompressImage

`cnt`

The number of samples to compress.

`inState`

A pointer to a 128-byte buffer from which the input state of the algorithm is read, or NIL. This buffer should be filled with zeros to initialize the algorithm.

`outState`

A pointer to a 128-byte buffer to which the output state of the algorithm is written, or NIL. This buffer may be the same as that specified by the `inState` parameter.

`numChannels`

The number of channels in the buffer pointed to by the `inBuffer` parameter.

`whichChannel`

The channel to compress when `numChannels` is greater than 1. The value of this parameter must be in the range 1 to `numChannels`.

Discussion

This function expands `cnt` packets of sound stored in the buffer specified by `inBuffer` and places the result in the buffer specified by `outBuffer`, whose size must be at least `cnt` packets times 1 byte per packet times 6, or `cnt * 6` bytes. If `numChannels` is greater than 1, then the compressed sound must be stored in interleaved format on a packet basis. The `Exp1to6` procedure works just like `Exp1to3` (I-342), but expands 1-byte packets rather than 2-byte packets.

Special Considerations

Because `Exp1to6` might allocate memory, you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 4. Macintosh Note: Not available in **CarbonLib**, but available if InterfaceLib 7.1 or later is installed.

Programming Info

C interface file: `Sound.h`

F

FCompressImage

Compresses a single-frame image that is currently stored as a pixel map structure, with added control over the compression process.

```
OSErr FCompressImage (
    PixMapHandle      src,
    const Rect        *srcRect,
    short              colorDepth,
    CodecQ             quality,
    CodecType          cType,
    CompressorComponent codec,
    CTabHandle         ctable,
    CodecFlags         flags,
    long               bufferSize,
    ICMFlushProcRecordPtr flushProc,
    ICMProgressProcRecordPtr progressProc,
    ImageDescriptionHandle desc,
    Ptr                data );
```

src

A handle to the image to be compressed. The image must be stored in a pixel map structure.

srcRect

A pointer to a rectangle defining the portion of the image to compress.

colorDepth

The depth at which the image is likely to be viewed. Compressors may use this as an indication of the color or grayscale resolution of the compressed image. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the `GetCodecInfo` (I-385) function.

quality

A constant (see below) that defines the desired compressed image quality.

cType

A compressor type. You must set this parameter to a valid compressor type constant.

codec

A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). Specifying a component instance may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation.

FCompressImage

`ctable`

A handle to a custom **color lookup table**. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the `colorDepth` parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

`flags`

Contains a flag (see below) that indicates whether or not the image was previously compressed.

`bufferSize`

The size of the buffer to be used by the data-unloading function specified by the `flushProc` parameter. If you have not specified a data-unloading function, set this parameter to 0.

`flushProc`

Points to an `ICMDDataProc` (III–2090) data-unloading callback. If there is not enough memory to store the compressed image, the compressor calls a function you provide that unloads some of the compressed data. If you have not provided a data-unloading callback, set this parameter to `NIL`. In this case, the compressor writes the entire compressed image into the memory location specified by the `data` parameter.

`progressProc`

Points to an `ICMPProgressProc` (III–2093) progress callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress callback, set this parameter to `NIL`. If you pass a value of `-1`, QuickTime provides a standard **progress function**.

`desc`

A handle that is to receive a formatted `ImageDescription` (IV–2285) structure. The Image Compression Manager resizes this handle for the returned `ImageDescription` (IV–2285) structure. Your application should store this image description with the compressed image data.

`data`

Points to a location to receive the compressed image data. It is your program's responsibility to make sure that this location can receive at least as much data as indicated by the `GetMaxCompressionSize` (I-420) function. If there is not sufficient memory to store the compressed image, you may choose to write the compressed data to mass storage during the compression operation. Use the `flushProc` parameter to identify your data-unloading function to the compressor. This pointer must contain a **32-bit clean** address. The Image Compression Manager places the actual size of the compressed image into the `dataSize` field of the `ImageDescription` (IV-2285) structure referenced by the `desc` parameter.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

quality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

codec Constants

`anyCodec`

The first compressor or decompressor of the specified type.

`bestSpeedCodec`

The fastest compressor or decompressor of the specified type.

`bestFidelityCodec`

The most accurate compressor or decompressor of the specified type.

`bestCompressionCodec`

The compressor that produces the smallest resulting data.

FCompressPicture

flags Constants

`codecFlagWasCompressed`

Indicates to the compressor that the image to be compressed has been compressed before. This information may be useful to compressors that can compensate for the image degradation that may otherwise result from repeated compression and decompression of the same image. Set this flag to 1 to indicate that the image was previously compressed. Set this flag to 0 if the image was not previously compressed.

Discussion

This function acts like `CompressImage` (I–113), but gives your application additional control over the parameters that guide the compression operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pixel Maps” (V–2789)

Related Java Methods

`quicktime.std.image.QTImage.fCompress()`

FCompressPicture

Compresses a single-frame image stored as a picture structure and places the result in another picture, with added control over the compression process.

```
OSErr FCompressPicture (
    PicHandle          srcPicture,
    PicHandle          dstPicture,
    short              colorDepth,
    CTabHandle         ctable,
    CodecQ             quality,
    short              doDither,
    short              compressAgain,
    ICMProgressProcRecordPtr progressProc,
    CodecType          cType,
    CompressorComponent codec );
```

`srcPicture`

A handle to the source image, stored as a picture.

`dstPicture`

A handle to the destination for the compressed image. The compressor resizes this handle for the result data.

`colorDepth`

The depth at which the image is to be compressed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the `GetCodecInfo` (I-385) function.

`ctable`

A handle to a custom **color lookup table**. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the `colorDepth` parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

`quality`

A constant that defines the desired compressed image quality.

`doDither`

A constant (see below) that indicates whether to **dither** the image. Use this parameter to indicate whether you want the image to be dithered when it is displayed on a lower-resolution screen.

`compressAgain`

Indicates whether to recompress compressed image data in the picture. Use this parameter to control whether any compressed image data that is in the source picture should be decompressed and then recompressed using the current parameters. Set the value of this parameter to `TRUE` to recompress such data. Set the value of the parameter to `FALSE` to leave the data as it is. Note that recompressing the data may have undesirable side effects, including image quality degradation.

FCompressPicture

`progressProc`

Points to an `ICMProgressProc` (III–2093) callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress callback, set this parameter to `NIL`. If you pass a value of `-1`, QuickTime provides a standard **progress function**.

`cType`

A compressor type. You must set this parameter to a valid compressor type constant; see “Codec Identifiers” (IV–2655). If the value passed in is 0, or ‘raw’, the resulting picture is not compressed and does not require QuickTime to be displayed.

`codec`

A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

quality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

Maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

doDither Constants

`defaultDither`

The **dithering** in the source file is to be respected.

`forceDither`

The specified image should be **dithered** whether the source uses dithering or not.

`suppressDither`

Dithering should not be used in any case. The ability to suppress **dithering** can be useful if, for example, you have a 32-bit, color JPEG image drawn into an 8-bit buffer with a custom color table from the image. In that case, dithering would not be necessary and probably not desirable, particularly if the buffer were to be compressed with the graphics compressor.

codec Constants

`anyCodec`

The first compressor or decompressor of the specified type.

`bestSpeedCodec`

The fastest compressor or decompressor of the specified type.

`bestFidelityCodec`

The most accurate compressor or decompressor of the specified type.

`bestCompressionCodec`

The compressor that produces the smallest resulting data.

Discussion

If a picture with multiple pixel maps and other graphical objects is passed, the pixel maps will be compressed individually and the other graphic objects will not be affected. `FCompressPicture` compresses only image data. Any other types of data in the picture, such as text, graphics primitives, and previously compressed images, are not modified in any way and are passed through to the destination picture. This function supports parameters governing image quality, compressor type, image depth, custom color tables, and **dithering**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pictures and PICT Files” (V-2790)

Related Java Methods

`quicktime.qd.Pict.fCompress()`

FCompressPictureFile

Compresses a single-frame image stored as a **picture file** and places the result in another picture file, with added control over the compression process.

```
OSErr FCompressPictureFile (
    short          srcRefNum,
    short          dstRefNum,
    short          colorDepth,
    CTabHandle     ctable,
    CodecQ         quality,
    short          doDither,
    short          compressAgain,
    ICMProgressProcRecordPtr progressProc,
    CodecType      cType,
    CompressorComponent codec );
```

srcRefNum

A file reference number for the source PICT file.

dstRefNum

A file reference number for the destination PICT file. Note that the compressor overwrites the contents of the file referred to by **dstRefNum**. You must open this file with write permissions. The destination file may be the same as the source file specified by the **srcRefNum** parameter.

colorDepth

The depth at which the image is to be compressed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor capability structure returned by the **GetCodecInfo** (I-385) function.

ctable

A handle to a custom **color lookup table**. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the **colorDepth** parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the **ctSeed** field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the **colorDepth** parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the **ctable**

parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

`quality`

A constant (see below) that defines the desired compressed image quality.

`doDither`

Indicates whether to **dither** the image. Use this parameter to indicate whether you want the image to be dithered when it is displayed on a lower-resolution screen. The following constants are available:

`compressAgain`

Indicates whether to recompress compressed image data in the picture. Use this parameter to control whether any compressed image data that is in the source picture should be decompressed and then recompressed using the current parameters. Set the value of this parameter to `TRUE` to recompress such data. Set the value of this parameter to `FALSE` to leave the data as it is. Note that recompressing the data may have undesirable side effects, including image quality degradation.

`progressProc`

Points to an `ICMProgressProc` (III–2093) callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress.

`cType`

A compressor type. You must set this parameter to a valid compressor type constant.

`codec`

A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

quality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

FCompressPictureFile

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a CodecQ field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

doDither Constants

`defaultDither`

The **dithering** in the source file is to be respected.

`forceDither`

The specified image should be **dithered** whether the source uses dithering or not.

`suppressDither`

Dithering should not be used in any case. The ability to suppress **dithering** can be useful if, for example, you have a 32-bit, color JPEG image drawn into an 8-bit buffer with a custom color table from the image. In that case, dithering would not be necessary and probably not desirable, particularly if the buffer were to be compressed with the graphics compressor.

codec Constants

`anyCodec`

The first compressor or decompressor of the specified type.

`bestSpeedCodec`

The fastest compressor or decompressor of the specified type.

`bestFidelityCodec`

The most accurate compressor or decompressor of the specified type.

`bestCompressionCodec`

The compressor that produces the smallest resulting data.

Discussion

This function compresses only image data. Any other types of data in the file, such as text, graphics primitives, and previously compressed images, are not modified in any way and are passed through to the destination **picture file**. This function supports parameters governing image quality, compressor type, image depth, custom color tables, and **dithering**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Working With Pictures and PICT Files” (V–2790)

FDecompressImage

Decompresses a single-frame image into a pixel map structure, with added control over the decompression process.

```
OSErr FDecompressImage (
    Ptr                data,
    ImageDescriptionHandle desc,
    PixMapHandle       dst,
    const Rect         *srcRect,
    MatrixRecordPtr    matrix,
    short              mode,
    RgnHandle          mask,
    PixMapHandle       matte,
    const Rect         *matteRect,
    CodecQ             accuracy,
    DecompressorComponent codec,
    long               bufferSize,
    ICMDataProcRecordPtr dataProc,
    ICMProgressProcRecordPtr progressProc );
```

data

Points to the compressed image data. If the entire compressed image cannot be stored at this location, your application may provide a data-loading function (see the discussion of the `dataProc` parameter to this function). This pointer must contain a **32-bit clean** address.

desc

A handle to the `ImageDescription` (IV–2285) structure that describes the compressed image.

dst

A handle to the pixel map where the decompressed image is to be displayed. Set the current graphics port to the port that contains this pixel map.

srcRect

A pointer to a rectangle defining the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the compressed image, which is defined by `(0,0)` and `((**desc).width, (**desc).height)`. If you want to decompress the entire source image, set this parameter to `NIL`. If the parameter

FDecompressImage

is NIL, the rectangle is set to the rectangle structure of the `ImageDescription` (IV-2285) structure.

`matrix`

Points to a matrix structure that specifies how to transform the image during decompression. You can use the matrix structure to translate or scale the image during decompression. If you do not want to apply such effects, set the `matrix` parameter to NIL.

`mode`

The transfer mode for the operation; see “Graphics Transfer Modes” (IV-2670).

`mask`

A handle to a clipping region in the destination coordinate system. If specified, the decompressor applies this mask to the destination image. If you do not want to mask bits in the destination, set this parameter to NIL.

`matte`

A handle to a pixel map that contains a blend matte. You can use the blend matte to cause the decompressed image to be blended into the destination pixel map. The matte can be defined at any supported pixel depth; the matte depth need not correspond to the source or destination depths. However, the matte must be in the coordinate system of the source image. If you do not want to apply a blend matte, set this parameter to NIL.

`matteRect`

A pointer to a rectangle defining a portion of the blend matte to apply. If you do not want to use the entire matte referred to by the `matte` parameter, use this parameter to specify a rectangle within that matte. If specified, this rectangle must be the same size as the rectangle specified by the `srcRect` parameter. If you want to use the entire matte, or if you are not providing a blend matte, set this parameter to NIL.

`accuracy`

A constant (see below) that defines the desired compression accuracy. For a good display of still images, you should specify at least `codecHighQuality`.

`codec`

A decompressor identifier. Specify a particular decompressor by setting this parameter to its identifier. Alternatively, you may use a special identifier (see below). Specifying a component instance may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation.

bufferSize

The size of the buffer to be used by the data-loading function specified by the dataProc parameter. If you have not specified a data-loading function, set this parameter to 0.

dataProc

Points to an ICMDataProc (III–2090) data-loading callback. If there is not enough memory to store the compressed image, the compressor calls a function you provide that loads more compressed data. If you have not provided a data-unloading callback, set this parameter to NIL. In this case, the compressor expects that the entire compressed image is in the memory location specified by the data parameter.

progressProc

Points to an ICMProgressProc (III–2093) progress callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress callback, set this parameter to NIL. If you pass a value of –1, QuickTime provides a standard **progress function**.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

accuracy Constants

codecMinQuality

The minimum valid value for a CodecQ field.

codecLowQuality

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

codecNormalQuality

Image reproduction of normal quality.

codecHighQuality

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

codecMaxQuality

The maximum standard value for a CodecQ field.

codecLosslessQuality

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

codec Constants

anyCodec

The first compressor or decompressor of the specified type.

FindCodec

`bestSpeedCodec`

The fastest compressor or decompressor of the specified type.

`bestFidelityCodec`

The most accurate compressor or decompressor of the specified type.

Discussion

This function gives your application greater control over the parameters that guide the decompression operation. If you find that you do not need this level of control, use `DecompressImage` (I–235). Note that this function is invoked through the `StdPix` (III–1912) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pixel Maps” (V–2789)

Related Java Methods

`quicktime.std.image.QTImage.fDecompress()`

FindCodec

Determines which of the installed compressors or decompressors has been chosen to field requests made by using one of the special compressor identifiers.

```
OSErr FindCodec (
    CodecType          cType,
    CodecComponent      specCodec,
    CompressorComponent *compressor,
    DecompressorComponent *decompressor );
```

`cType`

You must set this parameter to a valid compressor type constant; see “Codec Identifiers” (IV–2655).

`specCodec`

A special codec identifier value (see below).

`compressor`

A pointer to a field to receive the identifier for the compressor component. The Image Compression Manager returns the identifier of the compressor that meets the special characteristics you specify in the `specCodec` parameter. Note that this identifier may differ from the value of the field referred to by the `decompressor` field. The Image Compression Manager sets this field to 0 if it

cannot find a suitable compressor component. Set this parameter to `NIL` if you do not want this information.

`decompressor`

A pointer to a field to receive the identifier for the decompressor component. The Image Compression Manager returns the identifier of the decompressor that meets the special characteristics you specify in the `specCodec` parameter. Note that this identifier may differ from the value of the field referred to by the `compressor` field. The Image Compression Manager sets this field to 0 if it cannot find a suitable decompressor component. Set this parameter to `NIL` if you do not want this information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

specCodec Constants

`anyCodec`

The first compressor or decompressor of the specified type.

`bestSpeedCodec`

The fastest compressor or decompressor of the specified type.

`bestFidelityCodec`

The most accurate compressor or decompressor of the specified type.

`bestCompressionCodec`

The compressor that produces the smallest resulting data.

Discussion

Some Image Compression Manager functions allow you to specify a particular compressor component by its identifier. For example, you may use the `codec` parameter to `CompressSequenceBegin` (I–119) to specify a particular compressor to do the compression. The Image Compression Manager also supports several special identifiers (see `specCodec` Constants, above) that allow you to exert some control over the component for a given action without having to know its identifier.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting Information About Compressor Components” (V–2788)

Related Java Methods

`quicktime.std.image.Decompressor.findCodec()`,
`quicktime.std.image.Compressor.findCodec()`

FindNextComponent

See Also

See `GetCodecNameList` (I–386) for information on retrieving codec identifiers.

FindNextComponent

Returns the component identifier for the next registered component that meets the selection criteria specified by an application.

```
Component FindNextComponent (
    Component          aComponent,
    ComponentDescription *looking );
```

aComponent

The starting point for the search. Set this field to 0 to start the search at the beginning of the component list. If you are continuing a search, you can specify a component identifier previously returned by `FindNextComponent`. The function then searches the remaining components.

looking

A `ComponentDescription` (IV–2212) structure. Your application specifies the criteria for the component search in the fields of this structure. The Component Manager ignores fields that are set to 0. For example, if you set all the fields to 0, all components meet the search criteria. In this case, your application can retrieve information about all of the components that are registered in the system by repeatedly calling `FindNextComponent` and `GetComponentInfo` (I–389) until the search is complete. Similarly, if you set all fields to 0 except for the `componentManufacturer` field, the Component Manager searches all registered components for a component supplied by the manufacturer you specify. Note that the `FindNextComponent` function does not modify the contents of the `ComponentDescription` structure you supply. To retrieve detailed information about a component, you need to use the `GetComponentInfo` function to get the component description record for each returned component.

function result The component identifier of a component that meets the search criteria, or 0 if there are no more matching components.

Discussion

This function returns the component identifier of a component that meets the search criteria. It returns a function result of 0 when there are no more matching components. The following code sample illustrates its use:

```
// FindNextComponent coding example
// See "Discovering QuickTime," page 281
```

```

void findGraphicsExporterComponents()
{
    ComponentDescription cd;
    Component c = 0;

    cd.componentType = GraphicsExporterComponentType;
    cd.componentSubType = 0;
    cd.componentManufacturer = 0;
    cd.componentFlags = 0;
    cd.componentFlagsMask = graphicsExporterIsBaseExporter;

    while((c = FindNextComponent(&cd, c)) != 0) {
        ...           // add component c to the list.
    }
}

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Component Functions Used By Applications” (V–2781)

Related Java Methods

`quicktime.std.comp.ComponentIdentifier.find()`

F

FixExp2

Undocumented

```
Fixed FixExp2 (
    Fixed    src );
```

`src`

A fixed integer.

function result *Undocumented*

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

FixLog2

FixLog2

Undocumented

```
Fixed FixLog2 (  
    Fixed    src );
```

src

A fixed integer.

function result *Undocumented*

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

FixMulDiv

Undocumented

```
Fixed FixMulDiv (  
    Fixed    src,  
    Fixed    mul,  
    Fixed    divisor );
```

src

Undocumented

mul

Undocumented

divisor

Undocumented

function result *Undocumented*

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

FixPow

Undocumented

```
Fixed FixPow (
    Fixed    base,
    Fixed    exp );
```

base

Undocumented

exp

Undocumented

function result *Undocumented*

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

FlashMediaDoButtonActions

Performs actions attached to a specified button.

```
ComponentResult FlashMediaDoButtonActions (
    MediaHandler    mh,
    char            *path,
    long            buttonID,
    long            transition );
```

mh

The Toolbox's connection to your derived Flash media handler. You can obtain this reference from GetMediaHandler (I-432).

path

Specifies the path to the button to which the action is attached.

buttonID

The ID of the button.

transition

Sends a mouse transition message to the object and whatever Flash actions are associated with that transition on the object that should be performed. The values are specific Flash transition constants.

FlashMediaFrameLabelToMovieTime

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `Movies.h`

FlashMediaFrameLabelToMovieTime

Undocumented

```
ComponentResult FlashMediaFrameLabelToMovieTime (
    MediaHandler    mh,
    Ptr             theLabel,
    TimeValue       *movieTime );
```

mh

The Toolbox’s connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (I–432).

theLabel

Undocumented

movieTime

Undocumented

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

FlashMediaFrameNumberToMovieTime

Undocumented

```
ComponentResult FlashMediaFrameNumberToMovieTime (
    MediaHandler    mh,
```

FlashMediaGetDisplayedFrameNumber

```
long          flashFrameNumber,  
TimeValue    *movieTime );
```

mh

The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (I-432).

flashFrameNumber
Undocumented

movieTime
Undocumented

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

FlashMediaGetDisplayedFrameNumber

Undocumented

```
ComponentResult FlashMediaGetDisplayedFrameNumber (  
MediaHandler    mh,  
long            *flashFrameNumber );
```

mh

The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (I-432).

flashFrameNumber
Undocumented

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 4.

FlashMediaGetFlashVariable

Programming Info

C interface file: `Movies.h`

FlashMediaGetFlashVariable

Gets the value of a specified Flash action variable.

```
ComponentResult FlashMediaGetFlashVariable (
    MediaHandler    mh,
    char            *path,
    char            *name,
    Handle          *theVariableCStringOut );
```

mh

The Toolbox’s connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (I–432).

path

Specifies the path to the Flash button to which the variable is attached.

name

Specifies the name of the Flash variable.

theVariableCStringOut

A handle to the value of the Flash variable as a C string.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `Movies.h`

FlashMediaGetRefConBounds

Undocumented

```
ComponentResult FlashMediaGetRefConBounds (
    MediaHandler    mh,
    long            refCon,
    long            *left,
    long            *top,
```

```

    long          *right,
    long          *bottom );

```

mh

The Toolbox’s connection to your derived Flash media handler. You can obtain this reference from GetMediaHandler (I–432).

refCon

Undocumented

left

Undocumented

top

Undocumented

right

Undocumented

bottom

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

FlashMediaGetRefConID

Undocumented

```

ComponentResult FlashMediaGetRefConID (
    MediaHandler  mh,
    long          refCon,
    long          *refConID );

```

mh

The Toolbox’s connection to your derived Flash media handler. You can obtain this reference from GetMediaHandler (I–432).

refCon

Undocumented

FlashMediaGetSupportedSwfVersion

refConID

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

FlashMediaGetSupportedSwfVersion

Identifies the version of Flash that this version of QuickTime supports.

```
ComponentResult FlashMediaGetSupportedSwfVersion (  
    MediaHandler      mh,  
    unsigned char     *swfVersion );
```

mh

The Toolbox’s connection to your derived Flash media handler. You can obtain this reference from GetMediaHandler (I–432).

swfVersion

The version number of the most current version of Flash that this version of QuickTime supports.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: Movies.h

FlashMediaIDToRefCon

Undocumented

```
ComponentResult FlashMediaIDToRefCon (
```

```
MediaHandler    mh,  
long            refConID,  
long            *refCon );
```

mh

The Toolbox’s connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (I–432).

refConID

Undocumented

refCon

Undocumented

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

FlashMediaSetFlashVariable

Sets the specified Flash action variable to a value.

```
ComponentResult FlashMediaSetFlashVariable (  
    MediaHandler    mh,  
    char            *path,  
    char            *name,  
    char            *value,  
    Boolean          updateFocus );
```

mh

The Toolbox’s connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (I–432).

path

Specifies the path to the Flash button to which the variable is attached.

name

Specifies the name of the Flash variable.

value

Specifies the new value of the Flash variable.

FlashMediaSetPan

updateFocus

Pass TRUE if the focus is to be changed.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: Movies.h

FlashMediaSetPan

Undocumented

```
ComponentResult FlashMediaSetPan (
    MediaHandler    mh,
    short           xPercent,
    short           yPercent );
```

mh

Undocumented

xPercent

Undocumented

yPercent

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

FlashMediaSetZoom

Undocumented


```
ComponentResult FlashMediaSetZoom (
    MediaHandler    mh,
    short           factor );
```

mh

Undocumented

factor

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

FlashMediaSetZoomRect

Undocumented

```
ComponentResult FlashMediaSetZoomRect (
    MediaHandler    mh,
    long            left,
    long            top,
    long            right,
    long            bottom );
```

mh

Undocumented

left

Undocumented

top

Undocumented

right

Undocumented

bottom

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError

FlattenMovie

(I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

FlattenMovie

Creates a new movie file containing a specified movie.

```
void FlattenMovie (
    Movie          theMovie,
    long           movieFlattenFlags,
    const FSSpec   *theFile,
    OSType         creator,
    ScriptCode     scriptTag,
    long           createMovieFileFlags,
    short          *resId,
    ConstStr255Param resName );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

movieFlattenFlags

Contains flags (see below) that control the process of adding movie data to the new movie file. Set unused flags to 0.

theFile

A pointer to the file system specification for the movie file to be created.

creator

The creator value for the new file.

scriptTag

The script in which the movie file should be created. Set this parameter to the Script Manager constant `smSystemScript` to use the system script; set it to `smCurrentScript` to use the current script. See *Inside Macintosh: Text* (listed in the **bibliography**) for more information about scripts and script tags.

createMovieFileFlags

Contains flags (see below) that control file creation options.

resId

A pointer to a field that contains the **resource** ID number for the new resource. If the field referred to by the **resId** parameter is set to 0, the Movie Toolbox assigns a unique resource ID number to the new resource. The toolbox then returns the movie's resource ID number in the field referred to by the **resId** parameter. The Movie Toolbox assigns resource ID numbers sequentially, starting at 128. If the **resId** parameter is set to NIL, the Movie Toolbox assigns a unique resource ID number to the new resource and does not return that resource's ID value.

resName

Points to a character string with the name of the movie **resource**. If you set the **resName** parameter to NIL, the toolbox creates an unnamed resource.

function result You can access this function's error returns through **GetMoviesError** (I-492) and **GetMoviesStickyError** (I-494).

movieFlattenFlags Constants

flattenAddMovieToDataFork

Causes the movie to be placed in the data fork of the new movie file, as well as in the **resource** fork. You may use this flag to create movie files that are more easily moved to computer systems other than the Macintosh.

flattenDontInterleaveFlatten

Allows you to disable the Movie Toolbox's data storage optimizations. By default, the Movie Toolbox stores movie data in a format that is optimized for playback. Set this flag to 1 to disable these optimizations.

flattenActiveTracksOnly

Causes the Movie Toolbox to add only enabled movie tracks to the new movie file. Use **SetTrackEnabled** (III-1658) to enable and disable movie tracks.

flattenCompressMovieResource

Compresses the movie **resource** stored in the file's data fork, but not the movie resource stored in the resource fork on Mac OS systems.

flattenFSSpecPtrIsDataRefRecordPtr

Set this flag to 1 if the **FSSpec** pointer is a pointer to a **DataReferenceRecord** (IV-2233) structure. This capability enables you to flatten movies for devices other than file systems.

flattenForceMovieResourceBeforeMovieData

Set this flag when you are creating a fast start movie, to put the movie **resource** at the front of the resulting movie file.

FlattenMovieData

createMovieFileFlags Constants

createMovieFileDeleteCurFile

Indicates whether to delete an existing file. If you set this flag to 1, the Movie Toolbox deletes the file (if it exists) before creating the new movie file. If this flag is set to 0 and the file specified by the `fileSpec` parameter already exists, the Movie Toolbox uses the existing file. In this case, the toolbox ensures that the file has both a data and a **resource** fork. If this flag is not set, the data is appended to the file.

Discussion

The file created by `FlattenMovie` also contains all the data for the movie; that is, the Movie Toolbox resolves any data references and includes the corresponding movie data in the new movie file.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Saving Movies” (V–2715)

Related Java Methods

`quicktime.std.movies.Movie.flatten()`

See Also

To create multiplatform QuickTime movies, use `FlattenMovieData` (I–374) instead of `FlattenMovie`.

FlattenMovieData

Creates a new movie and a file that contains all the movie data.

```
Movie FlattenMovieData (
    Movie          theMovie,
    long           movieFlattenFlags,
    const FSSpec   *theFile,
    OSType         creator,
    ScriptCode     scriptTag,
    long           createMovieFileFlags );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

`movieFlattenFlags`

Contains flags (see below) that control the process of adding movie data to the new movie file. These flags affect how the toolbox adds movies to the new movie file later. Set unused flags to 0.

`theFile`

This parameter usually contains a pointer to the file system specification for the movie file to be created. In place of a `FSSpec` pointer, QuickTime lets you pass a pointer to a data reference structure to receive the flattened movie data.

`creator`

The creator value for the new file.

`scriptTag`

Contains constants (see below) that specify the script in which the movie file should be created. See *Inside Macintosh: Text* (listed in the **bibliography**) for more information about scripts and script tags.

`createMovieFileFlags`

Contains flags (see below) that control file creation options.

function result The identifier of the new movie. If the function could not create the movie, it sets this returned identifier to `NIL`.

movieFlattenFlags Constants

`flattenAddMovieToDataFork`

Causes the movie to be placed in the data fork of the new movie file. You may use this flag to create single-fork movie files, which can be more easily moved to computer systems other than Macintosh.

`flattenDontInterleaveFlatten`

Allows you to disable the Movie Toolbox's data storage optimizations. By default, the Movie Toolbox stores movie data in a format that is optimized for the storage device. Set this flag to 1 to disable these optimizations.

`flattenActiveTracksOnly`

Causes the Movie Toolbox to add only enabled movie tracks to the new movie file. Use `SetTrackEnabled` (III-1658), to enable and disable movie tracks.

`flattenCompressMovieResource`

Compresses the movie **resource** stored in the file's data fork, but not the movie resource stored in the resource fork on Mac OS systems.

`flattenFSSpecPtrIsDataRefRecordPtr`

Set this flag to 1 if the `FSSpec` pointer is a pointer to a `DataReferenceRecord` (IV-2233) structure. This capability enables you to flatten movies for devices other than file systems.

FlattenMovieData

`flattenForceMovieResourceBeforeMovieData`

Set this flag when you are creating a fast start movie, to put the movie **resource** at the front of the resulting movie file.

scriptTag Constants

`smSystemScript`

Use the system script.

`smCurrentScript`

Use the current script.

createMovieFileFlags Constants

`createMovieFileDeleteCurFile`

Indicates whether to delete an existing file. If you set this flag to 1, the Movie Toolbox deletes the file (if it exists) before creating the new movie file. If this flag is set to 0 and the file specified by the `fileSpec` parameter already exists, the Movie Toolbox uses the existing file. In this case, the toolbox ensures that the file has both a data and a **resource** fork. If this flag isn't set, the data is appended to the file.

Discussion

This function will take any movie and optionally make it self-contained, interleaved, and Fast Start. Unlike `FlattenMovie` (I-372), this function does not add the new movie **resource** to the new movie file; instead, `FlattenMovieData` returns the new movie to your application. Your application must dispose of the returned movie. You can use this function to create a single-fork movie file, by setting the `flattenAddMovieToDataFork` flag in the `movieFlattenFlags` parameter to 1. The Movie Toolbox then places the movie into the data fork of the movie file. Instead of flattening to a file, you can specify a data reference to flatten a movie to. The following two code samples show flattening a movie to a data location and to a file:

```
// FlattenMovieData used to flatten a movie to a data location
// create a 0-length handle
myHandle = NewHandleClear(mySize);
if (myHandle == NIL)
    goto bail;

// fill in the data reference record
myDataRefRec.dataRefType = HandleDataHandlerSubType;
myDataRefRec.dataRef = NewHandle(sizeof(Handle));
if (myDataRefRec.dataRef == NIL)
    goto bail;

*((Handle *)*(myDataRefRec.dataRef)) = myHandle;
```

```

myFlags = flattenFSSpecPtrIsDataRefRecordPtr;
myFile = (FSSpec *)&myDataRefRec;

// flatten the source movie into the handle
myMemMovie = FlattenMovieData(mySrcMovie, myFlags, myFile, 0L,
                               smSystemScript, 0L);

Movie aMovie;
aMovie = FlattenMovieData(theMovie,
    flattenAddMovieToDataFork |
    flattenForceMovieResourceBeforeMovieData,
    &theOutputFile, OSTypeConst('TVOD'), smSystemScript,
    createMovieFileDeleteCurFile | createMovieFileDontCreateResFile);

DisposeMovie(aMovie);

Movie aMovie;
aMovie = FlattenMovieData(theMovie,
    flattenAddMovieToDataFork,
    &theOutputFile, OSTypeConst('TVOD'), smSystemScript,
    createMovieFileDeleteCurFile | createMovieFileDontCreateResFile);

DisposeMovie(aMovie);

// FlattenMovieData used to flatten a movie to a Fast Start file
// See "Discovering QuickTime," page 257
myErr = OpenMovieFile(&myTempSpec, &myTempResRefNum, fsRdPerm);
if (myErr != noErr)
    goto bail;
myErr = NewMovieFromFile(&myTempMovie, myTempResRefNum, NIL, 0, 0, 0);
if (myErr != noErr)
    goto bail;
SetMovieProgressProc(myTempMovie, (MovieProgressUPP)-1, 0L);
// flatten the temporary file into a new movie file; put the movie
// resource first so that progressive downloading is possible
myPanoMovie = FlattenMovieData(
    myTempMovie,
    flattenDontInterleaveFlatten
    | flattenAddMovieToDataFork
    | flattenForceMovieResourceBeforeMovieData,

```

FracSinCos

```
&myDestSpec,  
FOUR_CHAR_CODE('TVOD'),  
smSystemScript,  
createMovieFileDeleteCurFile  
| createMovieFileDontCreateResFile);
```

Special Considerations

Through the `SetTrackLoadSettings` (III–1661) function, the Movie Toolbox allows you to set a movie’s preloading guidelines when you create the movie. The preload information is preserved when you save or flatten the movie (using either `FlattenMovie` or `FlattenMovieData`). In flattened movies, the tracks that are to be preloaded are stored at the start of the movie, rather than being interleaved with the rest of the movie data. This greatly improves preload performance because it is not necessary for the device storing the movie data to seek during retrieval of the data to be preloaded.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Saving Movies” (V–2715)

Related Java Methods

`quicktime.std.movies.Movie.flattenData()`

See Also

See `FlattenMovie` (I–372).

FracSinCos

Undocumented

```
Fract FracSinCos (  
    Fixed    degree,  
    Fract    *cosOut );
```

degree

Undocumented

cosOut

Undocumented

function result *Undocumented*

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

FSSpecToNativePathName

Extracts the native pathname from an FSSpec.

```
OSErr FSSpecToNativePathName (
    const FSSpec    *inFile,
    char            *outName,
    unsigned long    outLen,
    long            flags );
```

inFile

A pointer to a FSSpec.

outName

A pointer to a buffer to hold a C string.

outLen

Specifies the maximum size of the buffer in bytes, including the string terminator.

flags

Contains flags (see below) that control the conversion.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

flags Constants

kFullPathName

The full pathname should be returned.

kFileNameOnly

Only the part of the pathname corresponding to the file should be returned. This might be useful to return a string for a Windows title.

kDirectoryPathOnly

The full pathname up to and including the enclosing directory but not the filename should be returned. This can be useful to get a path for the enclosing directory that might be used to find related files in the same directory.

GDGetScale

Discussion

You can use this function to convert an `FSSpec` returned by QuickTime to a native pathname to be passed to the current operating system. It accepts an `FSSpec` and puts the equivalent pathname string into a buffer.

Special Considerations

You can set the output buffer size to `pathnameMaxBufferSize`. This value must also include the string terminator.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QTML.h`

G

GDGetScale

Returns the current scale of the given screen graphics device.

```
OSErr GDGetScale (
    GDHandle    gdh,
    Fixed       *scale,
    short       *flags );
```

gdh

A handle to a screen graphics device.

scale

Points to a fixed-point field to hold the scale result.

flags

Points to a short integer that returns the `status` parameter flags for the video driver. Currently, 0 is always returned in this field.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Controlling Hardware Scaling” (V–2718)

See Also

For the corresponding set function, see `GDSetScale` (I–382).

GDHasScale

Returns the closest possible scaling that a particular screen device can be set to in a given pixel depth.

```
OSErr GDHasScale (
    GDHandle    gdh,
    short       depth,
    Fixed       *scale );
```

gdh

A handle to a screen graphics device.

depth

The pixel depth of the screen device.

scale

Points to a fixed-point scale value. On input, this field should be set to the desired scale value. On output, this field will contain the closest scale available for the given depth. A scale of 0x10000 indicates normal size, 0x20000 indicates double size, and so on.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns scaling information for a particular screen device for a requested depth. This function allows you to query a screen device without actually changing it. For example, if you specify 0x20000 but the screen device does not support it, `GDHasScale` returns `noErr` and a scale of 0x10000. Because this function checks for a supported depth, your requested depth must be supported by the screen device.

Special Considerations

`GDHasScale` references the video driver through the graphics device structure.

Version Notes

Introduced in QuickTime 3 or earlier.

GDSetScale

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Controlling Hardware Scaling” (V–2718)

GDSetScale

Sets a screen graphics device to a new scale.

```
OSErr GDSetScale (
    GDHandle    gdh,
    Fixed       scale,
    short       flags );
```

gdh

A handle to a screen graphics device.

scale

A fixed-point scale value.

flags

Points to a short integer. It returns the status parameter flags for the video driver. Currently, 0 is always returned in this field.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Controlling Hardware Scaling” (V–2718)

See Also

For the corresponding get function, see `GDGetScale` (I–380).

GetBestDeviceRect

Selects the deepest of all available graphics devices, while treating 16-bit and 32-bit screens as having equal depth.

```
OSErr GetBestDeviceRect (
    GDHandle    *gdh,
    Rect        *rp );
```

gdh

A pointer to the handle of the rectangle for the chosen device. If you do not need the information in this parameter returned, specify NIL.

rp

A pointer to the rectangle that is adjusted for the height of the menu bar if the device is the main device. If you do not need the information in this parameter returned, specify NIL.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function does not center a rectangle on a device. Rather, it returns the rectangle for the best device. The following code sample illustrates its use:

```
// GetBestDeviceRect coding example
// See “Discovering QuickTime,” page 265

WindowPtr MakeMyWindow (void)
{
    WindowPtr    pMacWnd;
    Rect         rectWnd = {0, 0, 120, 160};
    Rect         rectBest;

    // figure out the best monitor for the window
    GetBestDeviceRect(NIL, &rectBest);

    // put the window in the top left corner of that monitor
    OffsetRect(&rectWnd, rectBest.left + 10, rectBest.top + 50);

    // create the window
    pMacWnd = NewCWindow(NIL, &rectWnd, "\pGrabber",
                        TRUE, noGrowDocProc, (WindowPtr)-1, TRUE, 0);

    // set the port to the new window
    SetPort(pMacWnd);

    return pMacWnd;
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

GetCallbackTimeBase

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Graphics Devices and Graphics Worlds” (V–2798)

GetCallbackTimeBase

Retrieves the time base of a **callback event**.

```
TimeBase GetCallbackTimeBase (  
    QTCallback    cb );
```

cb

The **callback event** for the operation. You obtain this value from the `NewCallback` (II–1053) function.

function result A pointer to a `TimeBaseRecord` (IV–2481) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Time Base Callback Functions” (V–2763)

Related Java Methods

```
quicktime.std.clocks.QTCallback.getTimeBase(),  
quicktime.std.clocks.TimeBase.fromQTCallback()
```

GetCallbackType

Retrieves a **callback event**’s type.

```
short GetCallbackType (  
    QTCallback    cb );
```

cb

The **callback event** for the operation. You obtain this value from `NewCallback` (II–1053).

function result The callback type constant (see below). If the high-order bit (defined by `callbackAtInterrupt`) of the returned value is set to 1, the event can be invoked at interrupt time.

Function Result Constants`callBackAtTime`

Indicates that the event is to be invoked at a specified time.

`callBackAtRate`

Indicates that the event is to be invoked when the rate for the time base reaches a specified value.

`callBackAtTimeJump`

Indicates that the event is to be invoked when the time base's time value changes by an amount that differs from its rate.

`callBackAtInterrupt`

Defines the bit of the returned value which indicates that the event can be invoked at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Time Base Callback Functions” (V–2763)

Related Java Methods

`quicktime.std.clocks.QTCallback.getType()`

GetCodecInfo

Returns information about a single compressor component.

```

OSErr GetCodecInfo (
    CodecInfo      *info,
    CodecType      cType,
    CodecComponent codec );

```

`info`

A pointer to a `CodecInfo` (IV–2202) structure. `GetCodecInfo` returns detailed information about the appropriate compressor component in this structure.

`cType`

Set this parameter to a valid compressor type constant; see “Codec Identifiers” (IV–2655). If you want information about any compressor of the type specified by this parameter, set the `codec` parameter to 0. The Image Compression Manager then returns information about the first compressor it finds of the type you have specified.

GetCodecNameList

`codec`

Set this parameter to the component identifier of the specific compressor for the request, or to 0 for any compressor. Component identifiers are available in the `CodecNameSpecList` (IV–2208) structure returned by `GetCodecNameList` (I–386).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting Information About Compressor Components” (V–2788)

Related Java Methods

`quicktime.std.image.CodecInfo.CodecInfo()`

GetCodecNameList

Retrieves a list of installed compressor components or types.

```
OSErr GetCodecNameList (
    CodecNameSpecListPtr    *list,
    short                    showAll );
```

`list`

A pointer to a field that is to receive a pointer to a `CodecNameSpecList` (IV–2208) structure. The Image Compression Manager creates the appropriate list and returns a pointer to that list in the field specified by this parameter.

`showAll`

A short integer that controls the contents of the list. Set this parameter to 1 to receive a list of the names of all installed compressor components; the returned list contains one entry for each installed compressor. Set this parameter to 0 to receive a list of the types of installed compressor components; the returned list contains one entry for each installed compressor type.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The `CodecType` data type defines a field in the `CodecNameSpec` (IV–2208) structure that identifies the compression method employed by a given compressor component. See “Codec Identifiers” (IV–2655). Apple Computer’s Developer Technical Support

group assigns these values so that they remain unique. These values correspond, in turn, to text strings that can identify the compression method to the user.

Special Considerations

GetCodecNameList creates the CodecNameSpecList structure in your application's current heap zone.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Getting Information About Compressor Components” (V–2788)

Related Java Methods

quicktime.std.image.CodecNameList.CodecNameList()

See Also

See CodecNameSpecList (IV–2208).

GetComponentIconSuite

Returns a handle to the component's icon suite (if it provides one).

```
OSErr GetComponentIconSuite (
    Component      aComponent,
    Handle          *iconSuite );
```

aComponent

A component instance. Your application obtains the component instance from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

iconSuite

A handle to the component's icon suite, if any. If the component has not provided an icon suite, `GetComponentIconSuite` returns NIL in this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns a handle to the component's icon suite. A component provides to the Component Manager the **resource ID** of its icon family in the optional extensions to the component resource. Your application is responsible for disposing of the returned icon suite handle.

GetComponentIndString

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Component Functions Used By Applications” (V–2781)

See Also

For information about Macintosh icon suites and icon families, see Chapter 5 of *Inside Macintosh: More Macintosh Toolbox* (listed in the **bibliography**).

GetComponentIndString

Gets a private component **resource** string.

```
OSErr GetComponentIndString (
    Component    aComponent,
    Str255       theString,
    short        strListID,
    short        index );
```

`aComponent`

A component instance. Your application obtains the component instance from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`theString`

The returned string.

`strListID`

The ID of the Macintosh 'STR#' **resource** that contains the string.

`index`

The index number of the string in the Macintosh 'STR#' **resource**.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You can use this function to get a single string from a Macintosh 'STR#' **resource** belonging to a component. The following code snippet illustrates its use:

```
Str255 str;
err = GetComponentIndString((Component)store->self, str, 128, 1);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

GetComponentInfo

Returns all of the registration information for a component.

```
OSErr GetComponentInfo (
    Component          aComponent,
    ComponentDescription *cd,
    Handle             componentName,
    Handle             componentInfo,
    Handle             componentIcon );
```

`aComponent`

A component identifier that specifies the component for the operation. Your application obtains a component identifier from `FindNextComponent` (I–360). Alternatively, you can supply a component instance. Your application obtains a component instance from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`cd`

A pointer to a `ComponentDescription` (IV–2212) structure, which contains information about the component’s type, subtype, and manufacturer.

`componentName`

A handle to the component’s name. You must allocate this handle before calling this function. Pass `NIL` if you don’t want this information.

`componentInfo`

A handle to the component’s information string. You must allocate this handle before calling this function. Pass `NIL` if you don’t want this information.

`componentIcon`

A handle to the component’s black-and-white icon. You must allocate this handle before calling this function. Pass `NIL` if you don’t want this information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Component Functions Used By Applications” (V–2781)

GetComponentInstanceA5

Related Java Methods

`quicktime.std.comp.ComponentIdentifier.getInfo()`

GetComponentInstanceA5

Obsolete.

```
long GetComponentInstanceA5 (
    ComponentInstance    aComponentInstance );
```

Version Notes

Introduced in QuickTime 3 or earlier. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: `Components.h`

See Also

For the corresponding set function, see `SetComponentInstanceA5` (III–1570).

GetComponentInstanceError

Returns the last error generated by a specific connection to a component.

```
OSErr GetComponentInstanceError (
    ComponentInstance    aComponentInstance );
```

`aComponentInstance`

A component instance. Your application obtains the component instance from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Component Functions Used By Applications” (V–2781)

See Also

For the corresponding set function, see `SetComponentInstanceError` (III–1571).

GetComponentInstanceStorage

Lets your component retrieve a handle to the memory associated with a component connection.

```
Handle GetComponentInstanceStorage (
    ComponentInstance    aComponentInstance );
```

aComponentInstance

A component instance. Your application obtains the component instance from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result A handle to the memory associated with a component connection.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

See Also

For the corresponding set function, see `SetComponentInstanceStorage` (III–1572).

GetComponentListModSeed

Determines if the list of registered components has changed.

```
long GetComponentListModSeed ( void );
```

function result The component registry seed number.

Discussion

This function lets you determine if the list of registered QuickTime components has changed. It returns the value of the component registry seed number for the component list. By comparing this value to values previously returned by this function, you can determine whether the component registry has changed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

GetComponentPublicResource

GetComponentPublicResource

Retrieves a component's public **resource**.

```
OSErr GetComponentPublicResource (
    Component    aComponent,
    OSType       resourceType,
    short        resourceID,
    Handle       *theResource );
```

aComponent

A component instance. Your application obtains the component instance from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131).

resourceType

The **resource**'s type.

resourceID

The **resource**'s ID.

theResource

A pointer to a handle to the **resource**. A component's public resources can be accessed without having to first open the component.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

Some types of components provide information about themselves through public component **resources**. For example, Movie Export components can each contain a public resource which indicates the kinds of QuickTime media types that may be exported. QuickTime visual effects components can each contain a public resource that describes the list of parameters that can be used to configure the effect.

Special Considerations

The **resources** returned by this function are cached. This means that successive calls to access the same resource are typically very fast.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Components.h`

GetComponentPublicResourceList

Obtains a single public **resource** from each component that matches a component description.

```
OSErr GetComponentPublicResourceList (
    OSType          resourceType,
    short           resourceID,
    long            flags,
    ComponentDescription *cd,
    GetMissingComponentResourceUPP missingProc,
    void            *refCon,
    void            *atomContainerPtr );
```

resourceType

Undocumented

resourceID

Undocumented

flags

Undocumented

cd

Undocumented

missingProc

A GetMissingComponentResourceProc (III–2084) callback.

refCon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

atomContainerPtr

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

flags Constants

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Components.h

GetComponentRefcon

GetComponentRefcon

Retrieves the value of the reference constant for your component.

```
long GetComponentRefcon (
    Component    aComponent );
```

aComponent

A component instance. Your application obtains the component instance from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result The component’s reference constant.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

See Also

For the corresponding set function, see `SetComponentRefcon` (III–1573).

GetComponentResource

Undocumented

```
OSErr GetComponentResource (
    Component    aComponent,
    OSType       resType,
    short        resID,
    Handle       *theResource );
```

aComponent

A component instance. Your application obtains the component instance from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

resType

Undocumented

resID

Undocumented

theResource

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

GetComponentTypeModSeed

Determines if the specified type of registered component has changed.

```
long GetComponentTypeModSeed (  
    OSType    componentType );
```

`componentType`

A four-character code that identifies the type of component. See “Component Identifiers” (IV–2657).

function result The component registration seed number for the specified component type.

Discussion

This function is similar to `GetComponentListModSeed` (I–391), but is specific to a single component type. This allows you to tell, for example, whether the registration of image decompressor ('imdc') components has changed regardless of other component changes. This function returns the value of the component registry seed number for the specified component type. By comparing this value to values previously returned by this function, you can determine whether the component registry for the specified type has changed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

GetComponentVersion

Returns a component’s version number.

```
long GetComponentVersion (  
    ComponentInstance  ci );
```

GetCompressedImageSize

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result The version number of the component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Component Functions Used By Applications” (V–2781)

GetCompressedImageSize

Determines the size, in bytes, of a compressed image.

```
OSErr GetCompressedImageSize (
    ImageDescriptionHandle desc,
    Ptr data,
    long bufferSize,
    ICMDataProcRecordPtr dataProc,
    long *dataSize );
```

desc

A handle to the `ImageDescription` (IV–2285) structure that defines the compressed image for the operation.

data

Points to the compressed image data. This pointer must contain a **32-bit clean** address.

bufferSize

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If you have not specified a data-loading function, set this parameter to 0.

dataProc

Points to an `ICMDataProc` (III–2090) callback. If the data stream is not all in memory when your program calls `GetCompressedImageSize`, the compressor calls a function you provide that loads more compressed data. If you have not provided a data-loading callback, set this parameter to `NIL`. In this case, the entire image must be in memory at the location specified by the `data` parameter.

dataSize

A pointer to a field that is to receive the size, in bytes, of the compressed image.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Most applications do not need to use this function because compressed images have a corresponding `ImageDescription` (IV–2285) structure with a size field. You only need to use this function if you do not have an image description structure associated with your data; for example, when you are taking a compressed image out of a movie one frame at a time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting Information About Compressed Data” (V–2787)

Related Java Methods

`quicktime.std.image.QTImage.getCompressedSize()`

GetCompressedPixmapInfo

Retrieves information about a compressed image.

```
OSErr GetCompressedPixmapInfo (
    PixmapPtr          pix,
    ImageDescriptionHandle *desc,
    Ptr                *data,
    long               *bufferSize,
    ICMDataProcRecord  *dataProc,
    ICMPProgressProcRecord *progressProc );
```

pix

Points to a structure that holds encoded compressed image data.

desc

A pointer to a field that is to receive a handle to the `ImageDescription` (IV–2285) structure that defines the compressed image. If you are not interested in this information, specify `NIL` in this parameter.

data

A pointer to a field that is to receive a pointer to the compressed image data. If the entire compressed image cannot be stored at this location, you can define a data-loading function for this operation. If you are not interested in this information, you may specify `NIL` in this parameter.

GetCompressionInfo

bufferSize

A pointer to a field that is to receive the size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If there is no data-loading function defined for this operation, this parameter is ignored. If you are not interested in this information, you may specify `NIL` in this parameter.

dataProc

A pointer to an `ICMDataProc` (III–2090) callback. If there is not enough memory to store the compressed image, the decompressor calls a function you provide that loads more compressed data. If there is no data-loading function for this image, the function sets the `dataProc` field in the function structure to `NIL`. If you are not interested in this information, specify `NIL` in this parameter.

progressProc

A pointer to an `ICMProgressProc` (III–2093) callback. During a decompression operation, the decompressor may occasionally call a function you provide in order to report its progress. If there is no **progress function** for this image, the function sets the `progressProc` field in the function structure to `NIL`. If you pass a value of `-1`, QuickTime provides a standard progress function. If you are not interested in progress information, specify `NIL` in this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With the `StdPix` Function” (V–2797)

See Also

For the corresponding set function, see `SetCompressedPixMapInfo` (III–1573).

GetCompressionInfo

Gets information about sound compression.

```
OSErr GetCompressionInfo (
    short          compressionID,
    OSType         format,
    short          numChannels,
    short          sampleSize,
    CompressionInfoPtr cp );
```

`compressionID`

The compression algorithm used on a data sample (see below).

`format`

The format of the compressed data (see below). This parameter is set to 0 if the `compressionID` parameter contains other than `fixedCompression`.

`numChannels`

The number of channels for compressed sound.

`sampleSize`

The size of each sample.

`cp`

A pointer to a `CompressionInfo` (IV-2222) structure.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

compressionID Constants

`notCompressed`

No compression.

`fixedCompression`

Fixed-size compression.

`variableCompression`

Variable-size compression.

format Constants

`'NONE'`

Not compressed.

`'MAC3'`

Compression format is **MACE** 3:1.

`'MAC6'`

Compression format is **MACE** 6:1.

`'ima4'`

Compression format is IMA 4:1.

Discussion

The AIFF-C Extended Common Chunk format does not contain a `compressionID` field. In this case (and when using `'snd '` **resources** where the `OSType` describing the compression format is known) you should always pass the constant `fixedCompression` in this parameter and the `OSType` value in the `format` parameter. The `format` field will then contain the `OSType` value representing the compression format. If you set the `compressionID` field in a compressed sound header to any value other than `fixedCompression`, then the `format` field is set to zero. If you pass

GetCompressionName

fixedCompression in the compressionID parameter you will need to pass a valid compression type here.

Special Considerations

There are some 'snd ' **resources** that do not store an OSType value in the format field of the compressed sound header to describe the compression format. You can still use GetCompressionInfo in this case by passing in the compressionID and passing 0 in the format parameter. The correct OSType will be returned in the format field of the CompressionInfo structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

Related Java Methods

quicktime.sound.CompressionInfo.get()

GetCompressionName

Describes a given compression format in a string that can be displayed to the user.

```
OSErr GetCompressionName (
    OSType      compressionType,
    Str255      compressionName );
```

compressionType

The compression format (see “Codec Identifiers” (IV–2655)).

compressionName

A string containing the compression format’s name.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The string returned by GetCompressionName can be used in pop-up menus and other user interface elements to allow the user to select a compression format.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

GetCompressionTime

Determines the estimated amount of time required to compress a given image.

```
OSErr GetCompressionTime (
    PixMapHandle      src,
    const Rect        *srcRect,
    short              colorDepth,
    CodecType          cType,
    CompressorComponent codec,
    CodecQ             *spatialQuality,
    CodecQ             *temporalQuality,
    unsigned long      *compressTime );
```

src

A handle to the source image. The source image must be stored in a pixel map structure. The compressor uses only the bit depth of this image to determine the compression time. You may set this parameter to NIL if you are interested only in information about quality settings.

srcRect

A pointer to a rectangle defining the portion of the source image to compress. You may set this parameter to NIL if you are interested only in information about quality settings. `GetCompressionTime` then uses the bounds of the source pixel map.

colorDepth

The depth at which the image is to be compressed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the `GetCodecInfo` (I-385) function.

cType

You must set this parameter to a valid compressor type constant; see “Codec Identifiers” (IV-2655).

codec

Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). You can also specify a component instance. This may be useful if you have previously

GetCompressionTime

set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation.

spatialQuality

A pointer to a field containing a constant (see below) that defines the desired compressed image quality. The Image Compression Manager sets this field to the closest actual quality that the compressor can achieve. If you are not interested in this information, pass `NIL` in this parameter.

temporalQuality

A pointer to a field containing a constant (see below) that defines the desired temporal quality. Use this value only with images that are part of image sequences. The Image Compression Manager sets this field to the closest actual quality that the compressor can achieve. If you are not interested in this information, pass `NIL` in this parameter.

compressTime

A pointer to a field to receive the compression time in milliseconds. If the compressor cannot determine the amount of time required to compress the image or if the compressor does not support this function, this field is set to 0. If you are not interested in this information, pass `NIL` in this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

codec Constants

anyCodec

The first compressor or decompressor of the specified type.

bestSpeedCodec

The fastest compressor or decompressor of the specified type.

bestFidelityCodec

The most accurate compressor or decompressor of the specified type.

bestCompressionCodec

The compressor that produces the smallest resulting data.

spatialQuality and temporalQuality Constants

codecMinQuality

The minimum valid value for a `CodecQ` field.

codecLowQuality

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

codecNormalQuality

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a CodecQ field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

This function allows you to verify that the quality settings you desire are supported by a given compressor component. You specify the compression characteristics, including compression type and quality, along with the image. The Image Compression Manager returns the maximum compression time for the specified image and parameters. Note that some compressors may not support this function. If the component you specify does not support this function, the Image Compression Manager returns a time value of 0.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting Information About Compressed Data” (V–2787)

Related Java Methods

`quicktime.std.image.QTImage.getCompressionTime()`

GetCSequenceDataRateParams

Obtains the data rate parameters previously set with `SetCSequenceDataRateParams` (III–1574).

```
OSErr GetCSequenceDataRateParams (
    ImageSequence      seqID,
    DataRateParamsPtr  params );
```

`seqID`

Contains the unique sequence identifier that was returned by `CompressSequenceBegin` (I–119).

GetCSequenceFrameNumber

params

Points to the data rate parameters structure associated with the sequence identifier specified in the `seqID` parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Constraining Compressed Data” (V–2795)

Related Java Methods

`quicktime.std.image.CSequence.getDataRateParams()`

See Also

For the corresponding set function, see `SetCSequenceDataRateParams` (III–1574).

GetCSequenceFrameNumber

Returns the current frame number of the specified sequence.

```
OSErr GetCSequenceFrameNumber (
    ImageSequence  seqID,
    long           *frameNumber );
```

seqID

Contains the unique sequence identifier that was returned by the `CompressSequenceBegin` (I–119) function.

frameNumber

A pointer to the current frame number of the sequence identified by the `seqID` parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Related Java Methods

`quicktime.std.image.CSequence.getFrameNumber()`

See Also

For the corresponding set function, see SetCSequenceFrameNumber (III–1576).

GetCSequenceKeyFrameRate

Determines the current **key frame** rate of a sequence.

```
OSErr GetCSequenceKeyFrameRate (
    ImageSequence  seqID,
    long           *keyFrameRate );
```

seqID

Contains the unique sequence identifier that was returned by CompressSequenceBegin (I–119).

keyFrameRate

A pointer to a long integer that specifies the maximum number of frames allowed between **key frames**. Key frames provide points from which a temporally compressed sequence may be decompressed.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Related Java Methods

quicktime.std.image.CSequence.getKeyFrameRate()

See Also

For the corresponding set function, see SetCSequenceKeyFrameRate (III–1577).

GetCSequenceMaxCompressionSize

Determines the maximum size an image will be after compression for a given compression sequence.

```
OSErr GetCSequenceMaxCompressionSize (
    ImageSequence  seqID,
    PixMapHandle   src,
    long           *size );
```

GetCSequencePrevBuffer

`seqID`

Contains the unique sequence identifier that was returned by the `CompressSequenceBegin` (I–119) function.

`src`

A handle to the source `PixMap` (IV–2332) structure. The compressor uses only the image’s size and pixel depth to determine the maximum size of the compressed image.

`size`

A pointer to a field to receive the maximum size, in bytes, of the compressed image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is similar to `GetMaxCompressionSize` (I–420), but operates on a compression sequence instead of requiring the application to pass individual parameters about the source image.

Special Considerations

Before calling `GetCSequenceMaxCompressionSize` you must have already started a compression sequence with `CompressSequenceBegin` (I–119)

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Compression Parameters” (V–2794)

Related Java Methods

`quicktime.std.image.CSequence.getMaxCompressionSize()`

See Also

See `GetMaxCompressionSize` (I–420).

GetCSequencePrevBuffer

Determines the location of the previous image buffer allocated by the compressor.

```
OSErr GetCSequencePrevBuffer (
    ImageSequence    seqID,
    GWorldPtr        *gworld );
```

`seqID`

Contains the unique sequence identifier that was returned by the `CompressSequenceBegin` (I–119) function.

`gworld`

A pointer to a field to receive a pointer to the `CGrafPort` (IV–2168) structure that describes the graphics world for the image buffer. You should not dispose of this graphics world; the returned pointer refers to a buffer that the Image Compression Manager is using. If the compressor has not allocated a buffer, `GetCSequencePrevBuffer` returns an error result code.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Note that this function only returns information about buffers that were allocated by the compressor. You cannot use this function to determine the location of a buffer you have provided.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Compression Parameters” (V–2794)

Related Java Methods

`quicktime.std.image.CSequence.prevBuffer()`,
`quicktime.qd.QDGraphics.fromCSequence()`

GetDataHandler

Retrieves the best data handler component to use with a given data reference.

```
Component GetDataHandler (
    Handle    dataRef,
    OSType    dataHandlerSubType,
    long      flags );
```

`dataRef`

A handle to the data reference. The type of information stored in the handle depends upon the data reference type specified by the `dataHandlerSubType` parameter.

GetDefaultOutputVolume

dataHandlerSubType

Identifies both the type of data reference and, by implication, the component subtype value assigned to the data handler components that operate on data references of that type.

flags

Contains flags (see below) that indicate the way in which you intend to use the data handler component. Note that not all data handlers necessarily support all services; for example, some data handler components may not support streaming writes. Set the appropriate flags to 1.

function result The best data handler component conforming to the parameters passed in.

flags Constants

kDataHCanRead

You intend to use the data handler component to read data.

kDataHCanWrite

You intend to use the data handler component to write data.

kDataHCanStreamingWrite

You intend to do streaming writes (as part of a movie-capture operation, for example).

Discussion

Once you have used this function to get information about the best data handler component for your data reference, you can open and use the component using Component Manager functions. If the function returns a value of `NIL`, the toolbox was unable to find an appropriate data handler component. For more information about the error that caused a return of `NIL`, call `GetMoviesError` (I-492).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Selecting Media Handlers” (V-2754)

Related Java Methods

`quicktime.std.movies.media.DataHandler.DataHandler()`

GetDefaultOutputVolume

Determines the default volume level of the current sound output device.

```
OSErr GetDefaultOutputVolume (
    long    *level );
```

level

On exit, the default volume level of the current sound output device. The values returned in the high and low words range from 0x0000 (silence) to 0x0100 (full volume).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier. Replaces the earlier function GetSoundVolume.

Programming Info

C interface file: Sound.h

See Also

For the corresponding set function, see SetDefaultOutputVolume (III–1582).

GetDSequenceImageBuffer

Determines the location of the offscreen image buffer allocated by a decompressor.

```
OSErr GetDSequenceImageBuffer (
    ImageSequence    seqID,
    GWorldPtr        *gworld );
```

seqID

Contains the unique sequence identifier that was returned by the DecompressSequenceBegin (I–238) function.

gworld

A pointer to a field to receive a pointer to the CGrafPort (IV–2168) structure describing the graphics world for the image buffer. You should not dispose of this graphics world; the returned pointer refers to a buffer that the Image Compression Manager is using. It is disposed of for you when the CDSequenceEnd (I–77) function is called. If the decompressor has not allocated a buffer, GetDSequenceImageBuffer returns an error result code.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

GetDSequenceMatrix

Programming Info

C interface file: `ImageCompression.h`
Programming summary: “Changing Sequence-Decompression Parameters”
(V–2795)

Related Java Methods

`quicktime.std.image.DSequence.getImageBuffer()`,
`quicktime.qd.QDGraphics.fromDSequenceImage()`

GetDSequenceMatrix

Gets the matrix that was specified for a decompression sequence by a call to `SetDSequenceMatrix` (III–1587), or that was set at `DecompressSequenceBegin` (I–238).

```
OSErr GetDSequenceMatrix (
    ImageSequence      seqID,
    MatrixRecordPtr    matrix );
```

seqID

Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (I–238).

matrix

Points to a matrix structure that specifies how to transform the image during decompression

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding set function, see `SetDSequenceMatrix` (III–1587).

GetDSequenceScreenBuffer

Determines the location of the offscreen screen buffer allocated by a decompressor.

```
OSErr GetDSequenceScreenBuffer (
    ImageSequence      seqID,
    GWorldPtr          *gworld );
```


seqID

The unique sequence identifier that was returned by the DecompressSequenceBegin (I-238) function.

gworld

A pointer to a field to receive a pointer to the CGrafPort (IV-2168) structure that describes the graphics world for the screen buffer. You should not dispose of this graphics world; the returned pointer refers to a buffer that the Image Compression Manager is using. It is disposed of for you when the CDSequenceEnd (I-77) function is called. If the decompressor has not allocated a buffer, GetDSequenceScreenBuffer returns an error result code.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Changing Sequence-Decompression Parameters” (V-2795)

Related Java Methods

```
quicktime.std.image.DSequence.getScreenBuffer(),
quicktime.qd.QDGraphics.fromDSequenceScreen()
```

GetFirstCallback

Returns the first **callback event** associated with a specified time base.

```
QTCallback GetFirstCallback (
    TimeBase    tb );
```

tb

Specifies the time base for the operation. Your component can obtain the time base reference from your ClockSetTimeBase (I-98) function or from the Movie Toolbox’s GetCallbackTimeBase (I-384) function.

function result A pointer to a CallbackRecord (IV-2165) structure. Your software can pass this structure to other functions, such as ClockRateChanged (I-97).

Version Notes

Introduced in QuickTime 3 or earlier.

GetGraphicsImporterForDataRef

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Toolbox Clock Support Functions” (V–2869)

GetGraphicsImporterForDataRef

Locates and opens a graphics importer component that can be used to draw the image from specified data reference.

```
OSErr GetGraphicsImporterForDataRef (
    Handle          dataRef,
    OSType          dataRefType,
    ComponentInstance *gi );
```

`dataRef`

The data reference to be drawn using a graphics importer component.

`dataRefType`

The type of data reference pointed to by the `dataRef` parameter; see “Data References” (IV–2661). For **alias**-based data references, the `dataRef` handle contains an `AliasRecord` and `dataRefType` is set to `rAliasType`.

`gi`

On return, contains a pointer to the `ComponentInstance` of the graphics importer. If no graphics importer can be found, this parameter will be set to `NIL`. If `GetGraphicsImporterForDataRef` is able to locate a graphics importer for the data reference, the returned graphics importer `ComponentInstance` will already be set up to draw from the specified data reference to the current port.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function tries to locate a graphics importer component for the specified data reference by checking the file extension (such as `.GIF` or `.JPG`), the Macintosh file type, and the MIME type of the file. The file extension is retrieved from the data reference by using `DataHGetFileName` (I–193) to call the data handler associated with the data reference. If a graphics importer cannot be found using the file’s type, file extension, or MIME type, `GetGraphicsImporterForDataRef` asks each graphics importer to validate the file, until it either finds an importer that can handle the file or exhausts the list of possible importers. This validation attempt can be quite time-consuming; to bypass it, call `GetGraphicsImporterForDataRefWithFlags` (I–413) instead.

Special Considerations

The caller of `GetGraphicsImporterForDataRef` is responsible for closing the returned `ComponentInstance` using `CloseComponent` (I–100). You must call `CloseComponent` when you are finished with the importer.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Obtaining a Graphics Importer Instance” (V–2878)

Related Java Methods

`quicktime.std.image.GraphicsImporter.GraphicsImporter()`

See Also

See `GetGraphicsImporterForDataRefWithFlags` (I–413).

GetGraphicsImporterForDataRefWithFlags

Locates and opens a graphics importer component for a data reference with flags that control the search process.

```
OSErr GetGraphicsImporterForDataRefWithFlags (
    Handle          dataRef,
    OSType          dataRefType,
    ComponentInstance *gi,
    long            flags );
```

`dataRef`

The data reference to be drawn using a graphics importer component.

`dataRefType`

The type of data reference pointed to by the `dataRef` parameter; see “Data References” (IV–2661). For **alias**-based data references, the `dataRef` handle contains an `AliasRecord` and `dataRefType` is set to `rAliasType`.

`gi`

On return, contains a pointer to the `ComponentInstance` of the graphics importer. If no graphics importer can be found, this parameter will be set to `NIL`. If `GetGraphicsImporterForDataRefWithFlags` is able to locate a graphics importer for the data reference, the returned graphics importer `ComponentInstance` will already be set up to draw from the specified data reference to the current port.

`flags`

Contains flags (see below) that control the graphics importer search process.

GetGraphicsImporterForFile

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

flags Constants

kDontUseValidateToFindGraphicsImporter

If a graphics importer cannot be found using the file’s type, file extension, or MIME type, give up immediately without attempting the slower process of asking each graphics importer to validate the file.

Discussion

This function tries to locate a graphics importer component for the specified data reference by checking the file extension (such as .GIF or .JPG), the Macintosh file type, and the MIME type of the file. The file extension is retrieved from the data reference by using DataHGetFileName (I–193) to call the data handler associated with the data reference. If a graphics importer cannot be found using the file’s type, file extension, or MIME type, this function asks each graphics importer to validate the file, until it either finds an importer that can handle the file or exhausts the list of possible importers. This validation attempt can be quite time-consuming; to bypass it, pass kDontUseValidateToFindGraphicsImporter in the flags parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Obtaining a Graphics Importer Instance” (V–2878)

Related Java Methods

quicktime.std.image.GraphicsImporter.GraphicsImporter()

See Also

See GetGraphicsImporterForDataRef (I–412).

GetGraphicsImporterForFile

Locates and opens a graphics importer component that can be used to draw a specified file.

```
OSErr GetGraphicsImporterForFile (  
    const FSSpec          *theFile,  
    ComponentInstance     *gi );
```

theFile

The file to be drawn using a graphics importer component.

`gi`

On return, contains a pointer to the `ComponentInstance` of the graphics importer. If no graphics importer can be found for the specified file, the `gi` will be set to `NIL`. If `GetGraphicsImporterForFile` is able to locate a graphics importer for the file, the returned graphics importer `ComponentInstance` will already be set up to draw the specified file to the current port.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function first tries to locate a graphics importer component for the specified file based on its file type. If it is unable to locate a graphics importer component based on the Macintosh file type, or the call is made on a non-Macintosh file, `GetGraphicsImporterForFile` will try to locate a graphics importer component based on the file extension (such as `.JPG` or `.GIF`). If a graphics importer cannot be found using the file’s type or extension, `GetGraphicsImporterForFile` asks each graphics importer to validate the file, until it either finds an importer that can handle the file or exhausts the list of possible importers. This validation attempt can be quite time-consuming. To bypass the validation attempt, call `GetGraphicsImporterForFileWithFlags` (I–416) instead. The following code sample illustrates the use of `GetGraphicsImporterForFile`:

```
// Get a graphics importer for the image file, determine the natural size
// of the image, and draw the image
// See “Discovering QuickTime,” page 274
void drawFile(const FSSpec *fss, const Rect *boundsRect)
{
    GraphicsImportComponent gi;
    GetGraphicsImporterForFile(fss, &gi);
    GraphicsImportSetBoundsRect(gi, boundsRect);
    GraphicsImportDraw(gi);
    CloseComponent(gi);
}
```

Special Considerations

The caller of `GetGraphicsImporterForFile` is responsible for closing the returned `ComponentInstance` using `CloseComponent` (I–100).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Obtaining a Graphics Importer Instance” (V–2878)

GetGraphicsImporterForFileWithFlags

Related Java Methods

`quicktime.std.image.GraphicsImporter.GraphicsImporter()`

See Also

See `GetGraphicsImporterForFileWithFlags` (I–416).

GetGraphicsImporterForFileWithFlags

Locates and opens a graphics importer component for a file with flags that control the search process.

```
OSErr GetGraphicsImporterForFileWithFlags (
    const FSSpec      *theFile,
    ComponentInstance *gi,
    long              flags );
```

theFile

The file to be drawn using a graphics importer component.

gi

On return, contains a pointer to the `ComponentInstance` of the graphics importer. If no graphics importer can be found for the specified file, the *gi* will be set to `NIL`. If `GetGraphicsImporterForFileWithFlags` is able to locate a graphics importer for the file, the returned graphics importer `ComponentInstance` will already be set up to draw the specified file to the current port.

flags

Contains flags (see below) that control the graphics importer search process.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`kDontUseValidateToFindGraphicsImporter`

If a graphics importer cannot be found using the file’s type or file extension, give up immediately without using the slower process of asking graphics importers to validate the file.

Discussion

This function first tries to locate a graphics importer component for the specified file based on its file type. If it is unable to locate a graphics importer component based on the Macintosh file type, or the call is made on a non-Macintosh file, `GetGraphicsImporterForFile` will try to locate a graphics importer component based on the file extension (such as `.JPG` or `.GIF`). If a graphics importer cannot be found using the file’s type or extension, `GetGraphicsImporterForFile` asks each graphics importer to validate the file, until it either finds an importer that can handle the file

or exhausts the list of possible importers. This validation attempt can be quite time-consuming. To bypass the validation attempt, pass `kDontUseValidateToFindGraphicsImporter` in the `flags` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

See Also

See `GetGraphicsImporterForFile` (I–414).

GetImageDescriptionCTable

Sets the custom color table for an image.

```
OSErr GetImageDescriptionCTable (
    ImageDescriptionHandle desc,
    CTabHandle             *ctable );
```

desc

A handle to the appropriate `ImageDescription` (IV–2285) structure.

ctable

A pointer to a field that is to receive a color table handle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns the color table for the image described by the `ImageDescription` (IV–2285) structure that is referred to by the `desc` parameter. The function correctly sizes the handle for the color table it returns.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pixel Maps” (V–2789)

Related Java Methods

```
quicktime.qd.ColorTable.fromImageDescription(),
quicktime.std.image.ImageDescription.getCTable()
```

See Also

For the corresponding set function, see `SetImageDescriptionCTable` (III–1593).

GetImageDescriptionExtension

GetImageDescriptionExtension

Returns a new handle with the data from a specified image description extension.

```
OSErr GetImageDescriptionExtension (
    ImageDescriptionHandle desc,
    Handle *extension,
    long idType,
    long index );
```

desc

A handle to the appropriate `ImageDescription` (IV–2285) structure.

extension

A pointer to a field to receive a handle to the returned data. The `GetImageDescriptionExtension` function returns the extended data for the image described by the `ImageDescription` (IV–2285) structure referred to by the *desc* parameter. The function correctly sizes the handle for the data it returns.

idType

Specifies the extension’s type value. Use this parameter to determine the data type of the extension. This parameter contains a four-character code, similar to an `OSType` field value.

index

Specifies the extension’s index value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function allows the application to get a copy of a specified image description extension. Note that each compressor type may have its own format for the extended data that is stored with an image. The extended data is similar in concept to the user data that applications can associate with QuickTime movies. Once you have added extended data to an image, you cannot delete it.

Special Considerations

The Image Compression Manager allocates a new handle and passes it back in the *extension* parameter. Your application should dispose of the handle when it is no longer needed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Image Descriptions” (V–2790)

Related Java Methods

quicktime.std.image.ImageDescription.getExtension()

GetMatrixType

Obtains information about a matrix.

```
short GetMatrixType (
    const MatrixRecord    *m );
```

m

Points to the MatrixRecord (IV–2304) structure for this operation.

function result A constant (see below) that defines the type of the matrix.

Function Result Constants

identityMatrixType

The specified matrix is an identity matrix.

translateMatrixType

The specified matrix defines a translation operation.

scaleMatrixType

The specified matrix defines a scaling operation.

scaleTranslateMatrixType

The specified matrix defines both a translation operation and a scaling operation.

linearMatrixType

The specified matrix defines a rotation, skew, or shear operation.

linearTranslateMatrixType

The specified matrix defines both a translation operation and a rotation, skew, or shear operation.

perspectiveMatrixType

The specified matrix defines a perspective (nonlinear) operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Managing Matrices” (V–2764)

Related Java Methods

quicktime.std.image.Matrix.getType()

GetMaxCompressionSize

GetMaxCompressionSize

Determines the maximum size an image will be after compression.

```
OSErr GetMaxCompressionSize (
    PixMapHandle      src,
    const Rect        *srcRect,
    short              colorDepth,
    CodecQ             quality,
    CodecType          cType,
    CompressorComponent codec,
    long               *size );
```

src

A handle to the source image. The source image must be stored in a pixel map structure. The compressor uses only the image's size and pixel depth to determine the maximum size of the compressed image.

srcRect

A pointer to a rectangle defining the portion of the source image that is to be compressed. You may set this parameter to NIL if you are interested only in information about quality settings. `GetCompressionTime` (I-401) then uses the bounds of the source pixel map.

colorDepth

The depth at which the image is to be compressed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by `GetCodecInfo` (I-385).

quality

A constant (see below) that defines the desired compressed image quality.

cType

You must set this parameter to a valid compressor type constant; see “Codec Identifiers” (IV-2655).

codec

A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). You can also specify a component instance. This may be useful if you have previously set some parameter on a specific instance of a

codec field and want to make sure that the specified instance is used for that operation.

size

A pointer to a field to receive the size, in bytes, of the compressed image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

quality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

codec Constants

`anyCodec`

The first compressor or decompressor of the specified type.

`bestSpeedCodec`

The fastest compressor or decompressor of the specified type.

`bestFidelityCodec`

The most accurate compressor or decompressor of the specified type.

`bestCompressionCodec`

The compressor that produces the smallest resulting data.

Discussion

This function returns the maximum resulting size for the specified image and parameters. Your application may then use this information to allocate memory for the compression operation. The following code sample illustrates its use:

```
// GetMaxCompressionSize coding example
// See “Discovering QuickTime,” page 286
```

GetMaxCompressionSize

```
PicHandle GetQTCompressedPict (PixMapHandle hpmImage)
{
    long                lMaxCompressedSize = 0;
    Handle              hCompressedData = NIL;
    Ptr                 pCompressedData;
    ImageDescriptionHandle hImageDesc = NIL;
    OSErr               nErr;
    PicHandle           hpicPicture = NIL;
    Rect                rectImage = (**hpmImage).bounds;
    CodecType           dwCodecType = kJPEGCodecType;
    CodecComponent       codec = (CodecComponent)anyCodec;
    CodecQ               dwSpatialQuality = codecNormalQuality;
    short               nDepth = 0;          // let ICM choose depth

    nErr = GetMaxCompressionSize(hpmImage, &rectImage, nDepth,
                                dwSpatialQuality,
                                dwCodecType,
                                (CompressorComponent)codec,
                                &lMaxCompressedSize);

    if (nErr != noErr)
        return NIL;

    hImageDesc = (ImageDescriptionHandle)NewHandle(4);
    hCompressedData = NewHandle(lMaxCompressedSize);
    if ((hCompressedData != NIL) && (hImageDesc != NIL)) {
        MoveHHI(hCompressedData);
        HLock(hCompressedData);
        pCompressedData = StripAddress(*hCompressedData);

        nErr = CompressImage(hpmImage,
                             &rectImage,
                             dwSpatialQuality,
                             dwCodecType,
                             hImageDesc,
                             pCompressedData);

        if (nErr == noErr) {
            ClipRect(&rectImage);
            hpicPicture = OpenPicture(&rectImage);
            nErr = DecompressImage(pCompressedData,
                                  hImageDesc,
```

```

        hpmImage,
        &rectImage,
        &rectImage,
        srcCopy,
        NIL);

    ClosePicture();
}

    if (theErr || (GetHandleSize((Handle)hpicPicture) ==
        sizeof(Picture))) {
        KillPicture(hpicPicture);
        hpicPicture = NIL;
    }
}

if (hImageDesc != NIL)
    DisposeHandle((Handle)hImageDesc);

if (hCompressedData != NIL)
    DisposeHandle(hCompressedData);

return hpicPicture;
}

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Getting Information About Compressed Data” (V-2787)

Related Java Methods

quicktime.std.image.QTImage.getMaxCompressionSize()

GetMaxLoadedTimeInMovie

When a movie is being progressively downloaded, returns the duration of the part of a movie that has already been downloaded.

```

OSErr GetMaxLoadedTimeInMovie (
    Movie          theMovie,

```

GetMediaCreationTime

```
TimeValue    *time );
```

theMovie

The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

time

The duration of the part of a movie that has already been downloaded. This time value is expressed in the movie’s time coordinate system. If all of a movie has been downloaded, this parameter returns the duration of the entire movie.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The Movie Toolbox creates a time table for a movie when either `QTMovieNeedsTimeTable` (II–1202) or `GetMaxLoadedTimeInMovie` is called for the movie, but the time table is used only by the toolbox and is not accessible to applications. The toolbox disposes of the time table when the download is complete.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Download Control” (V–2772)

Related Java Methods

```
quicktime.std.movies.Movie.maxLoadedTimeInMovie()
```

GetMediaCreationTime

Returns the creation date and time stored in a media.

```
long GetMediaCreationTime (
    Media    theMedia );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result The media’s creation date and time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Determining Movie Creation and Modification Time” (V–2741)

Related Java Methods

`quicktime.std.movies.media.Media.getCreationTime()`

GetMediaDataHandler

Determines a media’s data handler.

```
DataHandler GetMediaDataHandler (
    Media    theMedia,
    short    index );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

index

Identifies the data reference. You provide the index value that corresponds to the data reference for which you want to retrieve the data handler. You must set this parameter to 1.

function result A data handler component instance.

Special Considerations

QuickTime normally takes care of selecting data handlers for media. Your application should not need to call this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Selecting Media Handlers” (V–2754)

Related Java Methods

`quicktime.std.movies.media.Media.getDataHandler()`

See Also

For the corresponding set function, see `SetMediaDataHandler` (III–1594).

GetMediaDataHandlerDescription

GetMediaDataHandlerDescription

Retrieves information about a media's data handler.

```
void GetMediaDataHandlerDescription (
    Media      theMedia,
    short      index,
    OSType     *dhType,
    Str255     creatorName,
    OSType     *creatorManufacturer );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

index

Identifies the data reference. You provide the index value that corresponds to the data reference for which you want to retrieve the data handler description. You must set this parameter to 1.

dhType

A pointer to a field of data type `OSType`. The Movie Toolbox returns the data handler type identifier. This value indicates the type of data reference supported by this data handler. This value also corresponds to the component subtype specified for the data handler component. All QuickTime data references have a type value of 'alis'. If you don't want to receive this information, set this parameter to `NIL`.

creatorName

Points to a string. The Movie Toolbox returns the name of the data handler's creator. If you don't want to receive this information, set this parameter to `NIL`.

creatorManufacturer

A pointer to a long integer. The Movie Toolbox returns the 4-byte value that identifies the manufacturer of the component. If you don't want to retrieve this information, set this parameter to `NIL`.

function result You can access this function's error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Selecting Media Handlers" (V–2754)

Related Java Methods

```
quicktime.std.movies.media.Media.getDataHandlerDescription()
```

GetMediaDataRef

Returns a copy of a specified data reference.

```
OSErr GetMediaDataRef (
    Media      theMedia,
    short      index,
    Handle     *dataRef,
    OSType     *dataRefType,
    long       *dataRefAttributes );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

index

The index value that corresponds to the data reference. It must be less than or equal to the value that is returned by `GetMediaDataRefCount` (I–428).

dataRef

A pointer to a field that is to receive a handle to the data reference. The media handler returns a handle to information that identifies the file that contains this media's data. The type of information stored in that handle depends upon the value of the `dataRefType` parameter. If the function cannot locate the specified data reference, the handler sets this returned value to NIL. Set the `dataRef` parameter to NIL if you are not interested in this information.

dataRefType

A pointer to a field that is to receive the type of data reference. If the data reference is an **alias**, the media handler sets this value to 'alis'. Set the `dataRefType` parameter to NIL if you are not interested in this information.

dataRefAttributes

A pointer to a field that is to receive the reference's attribute flags (see below). Unused flags are set to 0.

function result

You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

GetMediaDataRefCount

dataRefAttributes Constants

dataRefSelfReference

Indicates whether the data reference refers to the movie **resource**'s data file. If this flag is set to 1, the data reference identifies media data that is stored in the same file as the movie resource.

dataRefWasNotResolved

Indicates whether the Movie Toolbox resolved the data reference. If this flag is set to 1, the Movie Toolbox could not resolve the data reference. For example, the toolbox may be unable to resolve data references because the required storage device is unavailable at the time a movie is loaded. If the data reference is unresolved, the Movie Toolbox disables the corresponding track.

Discussion

Use this function to retrieve information about a data reference. For example, you might want to verify the condition of a movie's data references after loading the movie from its movie file. You could use this function to check each data reference.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Working With Data References" (V-2740)

Related Java Methods

```
quicktime.std.movies.media.Media.getDataRef(),  
quicktime.std.movies.media.DataRef.fromMedia()
```

See Also

For the corresponding set function, see `SetMediaDataRef` (III-1595).

GetMediaDataRefCount

Determines the number of data references in a media.

```
OSErr GetMediaDataRefCount (  
    Media    theMedia,  
    short    *count );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II-1120) and `GetTrackMedia` (I-535).

count

A pointer to a field that is to receive the number of data references in the media.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Data References” (V–2740)

Related Java Methods

`quicktime.std.movies.media.Media.getDataRefCount()`

GetMediaDataSize

Determines the size, in bytes, of the sample data in a media segment.

```
long GetMediaDataSize (
    Media      theMedia,
    TimeValue  startTime,
    TimeValue  duration );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

startTime

A time value specifying the starting point of the segment.

duration

A time value that specifies the duration of the segment.

function result The size, in bytes, of the sample data in the defined media segment.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Samples” (V–2742)

Related Java Methods

`quicktime.std.movies.media.Media.getDataSize()`

GetMediaDataSize64

GetMediaDataSize64

Provides a 64-bit version of GetMediaDataSize (I–429).

```
OSErr GetMediaDataSize64 (
    Media      theMedia,
    TimeValue  startTime,
    TimeValue  duration,
    wide       *dataSize );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as NewTrackMedia (II–1120) and GetTrackMedia (I–535).

startTime

A time value specifying the starting point of the segment.

duration

A time value that specifies the duration of the segment.

dataSize

The size, in bytes, of the sample data in the defined media segment.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The only difference between this function and GetMediaDataSize (I–429) is that the *dataSize* parameter returns a 64-bit integer instead of the function returning a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

GetMediaDuration

Returns the duration of a media.

```
TimeValue GetMediaDuration (
    Media      theMedia );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result The media’s duration.

Discussion

The following code sample illustrates the use of `GetMediaDuration`:

```
// GetMediaDuration coding example
// See “Discovering QuickTime,” page 89
void CreateMyVideoTrack (Movie movie)
{
    Track    track;
    Media    media;
    Rect     rect = {0, 0, 100, 320};

    track = NewMovieTrack(movie,
        FixRatio(rect.right, 1),
        FixRatio(rect.bottom, 1),
        kNoVolume);

    media = NewTrackMedia(track,
        VideoMediaType,
        600,                                // video time scale
        NIL, NIL);

    BeginMediaEdits(media);
    MyAddVideoSamplesToMedia(media, &rect);    // assemble data
    EndMediaEdits(media);
    InsertMediaIntoTrack(track,
        0,                                    // track start time
        0,                                    // media start time
        GetMediaDuration(media),
        kFix1);                                // normal speed
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Time” (V–2735)

GetMediaHandler

Related Java Methods

`quicktime.std.movies.media.Media.getDuration()`

GetMediaHandler

Obtains a reference to a media handler component.

```
MediaHandler GetMediaHandler (  
    Media    theMedia );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result A media handler component instance.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Selecting Media Handlers” (V–2754)

Related Java Methods

```
quicktime.std.qtcomponents.TimeCoder.fromMedia(),  
quicktime.std.movies.media.Media.getHandler(),  
quicktime.std.movies.media.MusicMedia.getMusicHandler(),  
quicktime.std.movies.media.VideoMedia.getVideoHandler(),  
quicktime.std.movies.media.SoundMedia.getSoundHandler(),  
quicktime.std.movies.media.SpriteMedia.getSpriteHandler(),  
quicktime.std.movies.media.ThreeDMedia.getThreeDHandler(),  
quicktime.std.movies.media.TextMedia.getTextHandler(),  
quicktime.std.movies.media.TimeCodeMedia.getTimeCodeHandler()
```

See Also

See `MediaSetGraphicsMode` (II–892). For the set function corresponding to `GetMediaHandler`, see `SetMediaHandler` (III–1598).

GetMediaHandlerDescription

Retrieves information about a media handler.

```
void GetMediaHandlerDescription (
```

```
Media      theMedia,
OSType     *mediaType,
Str255     creatorName,
OSType     *creatorManufacturer );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

mediaType

A pointer to a field in which the Movie Toolbox returns the media type identifier (see below). This value indicates the type of media supported by this media handler. This value also corresponds to the component subtype specified for the media handler component. If you don't want to receive this information, set the `mediaType` parameter to `NIL`.

creatorName

Points to a string. The Movie Toolbox returns the name of the media handler's creator. If you don't want to receive this information, set this parameter to `NIL`.

creatorManufacturer

A pointer to a long integer. The Movie Toolbox returns the 4-byte value that identifies the manufacturer of the component. If you don't want to retrieve this information, set this parameter to `NIL`.

function result You can access this function's error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

mediaType Constants

```
VideoMediaType
    Video media.

SoundMediaType
    Sound media.

TextMediaType
    Text media.
```

Discussion

The following code sample illustrates the use of `GetMediaHandlerDescription`:

```
// GetMediaHandlerDescription coding example
// See "Discovering QuickTime," page 363
Movie      movie1;
TimeValue  101dDuration;
Movie      movie2;
long       1Index, 1OrigTrackCount, 1ReferenceIndex;
```

GetMediaHandlerDescription

```
Track          track, trackSprite;

// get the first track in original movie and position at the start
trackSprite = GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);

// remove all tracks except video in modifier movie
for (lIndex = 1; lIndex <= GetMovieTrackCount(movie2); lIndex++) {
    Track          track = GetMovieIndTrack(movie2, lIndex);
    OSType          dwType;

    GetMediaHandlerDescription(GetTrackMedia(track),
                              &dwType, NIL, NIL);
    if (dwType != VideoMediaType) {
        DisposeMovieTrack(track);
        lIndex--;
    }
}

// add the modifier track to original movie
lOldDuration = GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);

// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, lOldDuration,
                  GetMovieDuration(movie1) - lOldDuration);

// associate the modifier track with the original sprite track
track = GetMovieIndTrack(movie1, lOrigTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                  &lReferenceIndex);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Selecting Media Handlers” (V-2754)

Related Java Methods

`quicktime.std.movies.media.Media.getHandlerDescription()`

GetMediaInputMap

Returns a copy of the input map associated with a specified media.

```
OSErr GetMediaInputMap (
    Media          theMedia,
    QTAtomContainer *inputMap );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

inputMap

The media input map for this operation. You must dispose of the map referred to by this parameter when you are done with it using `QTDisposeAtomContainer` (II–1164).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Use this function to specify the media you want to get so you can modify its input map, as illustrated below:

```
// GetMediaInputMap coding example
// See “Discovering QuickTime,” page 365
#define kImageIndexToOverride      1

Movie          movie1, movie2;
long           lReferenceIndex, lImageIndexToOverride;
Track          trackSprite;
QTAtomContainer qtacInputMap;
QTAtom         lInputAtom;
OSType         dwInputType;
Media          mediaSprite;

// get the sprite media’s input map
mediaSprite = GetTrackMedia(trackSprite);
GetMediaInputMap(mediaSprite, &qtacInputMap);

// add an atom for a modifier track
QTInsertChild(qtacInputMap, kParentAtomIsContainer, kTrackModifierInput,
              lReferenceIndex, 0, 0, NIL, &lInputAtom);
```

GetMediaLanguage

```
// add a child atom to specify the input type
dwInputType = kTrackModifierTypeImage;
QTInsertChild(qtacInputMap, lInputAtom, kTrackModifierType, 1, 0,
              sizeof(dwInputType), &dwInputType, NIL);

// add a second child atom to specify index of image to override
lImageIndexToOverride = EndianS16_NtoB(kImageIndexToOverride);
QTInsertChild(qtacInputMap, lInputAtom, kSpritePropertyImageIndex, 1, 0,
              sizeof(lImageIndexToOverride), &lImageIndexToOverride, NIL);

// update the sprite media's input map
SetMediaInputMap(mediaSprite, qtacInputMap);
QTDisposeAtomContainer(qtacInputMap);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Manipulating Media Input Maps” (V–2753)

Related Java Methods

```
quicktime.std.movies.media.Media.getInputMap(),
quicktime.std.movies.AtomContainer.fromMediaInput()
```

See Also

For the corresponding set function, see `SetMediaInputMap` (III–1599).

GetMediaLanguage

Returns a media’s localized language or **region code**.

```
short GetMediaLanguage (
    Media    theMedia );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result The media’s language or **region code**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Alternate Tracks” (V–2738)

Related Java Methods

`quicktime.std.movies.media.Media.getLanguage()`

See Also

For the corresponding set function, see `SetMediaLanguage` (III–1600). See *Inside Macintosh: Text* (listed in the **bibliography**) for more information about language and **region codes**.

GetMediaModificationTime

Returns a media’s modification date and time.

```
long GetMediaModificationTime (  
    Media    theMedia );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result The media’s modification date and time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Determining Movie Creation and Modification Time” (V–2741)

Related Java Methods

`quicktime.std.movies.media.Media.getModificationTime()`

GetMediaNextInterestingTime

Searches for times of interest in a media.

```
void GetMediaNextInterestingTime (  
    Media    theMedia,  
    short    interestingTimeFlags,  
    TimeValue time,  
    Fixed    rate,
```

GetMediaNextInterestingTime

```
TimeValue    *interestingTime,  
TimeValue    *interestingDuration );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

interestingTimeFlags

Contains flags (see below) that determine the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit` or `nextTimeSyncSample` flags to 1. Set unused flags to 0.

time

Specifies a time value that establishes the starting point for the search. This time value must be expressed in the media's time scale.

rate

The search direction. Negative values cause the Movie Toolbox to search backward from the starting point specified in the `time` parameter. Other values cause a forward search.

interestingTime

A pointer to a time value. The Movie Toolbox returns the first time value it finds that meets the search criteria specified in the `flags` parameter. This time value is in the media's time scale. If there are no times that meet the search criteria you specify, the Movie Toolbox sets this value to –1. Set this parameter to `NIL` if you are not interested in this information.

interestingDuration

A pointer to a time value. The Movie Toolbox returns the duration of the interesting time. This time value is in the media's time coordinate system. Set this parameter to `NIL` if you don't want this information; this lets the function work faster.

function result You can access this function's error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

interestingTimeFlags Constants

nextTimeMediaSample

Set this flag to 1 to search for the next sample.

nextTimeMediaEdit

Set this flag to 1 to search for the next group of samples.

nextTimeSyncSample

Set this flag to 1 to search for the next **sync sample**.

nextTimeEdgeOK

Set this flag to 1 to accept information about elements that begin or end at the time specified by the `time` parameter. When this flag is set the function returns valid information about the beginning and end of a media.

Discussion

Some compression algorithms conserve space by eliminating duplication between consecutive frames in a sample. They do this by deriving frames from **sync samples**, which don't rely on preceding frames for content.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Finding Interesting Times” (V-2735)

Related Java Methods

`quicktime.std.movies.media.Media.getNextInterestingTime()`

GetMediaPlayHints

Undocumented

```
void GetMediaPlayHints (
    Media    theMedia,
    long     *flags );
```

`theMedia`

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II-1120) and `GetTrackMedia` (I-535).

`flags`

A pointer to flags (see below).

function result *Undocumented*

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

GetMediaPreferredChunkSize

Related Java Methods

`quicktime.std.movies.media.Media.getPlayHints()`

See Also

For the corresponding set function, see `SetMediaPlayHints` (III–1601).

GetMediaPreferredChunkSize

Retrieves the maximum chunk size for a media.

```
OSErr GetMediaPreferredChunkSize (  
    Media      theMedia,  
    long       *maxChunkSize );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

maxChunkSize

Specifies a field to receive the maximum chunk size, in bytes.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Adding Samples to Media Structures” (V–2752)

Related Java Methods

`quicktime.std.movies.media.Media.getPreferredChunkSize()`

See Also

For the corresponding set function, see `SetMediaPreferredChunkSize` (III–1603).

GetMediaPropertyAtom

Retrieves the property atom container of a media handler.

```
OSErr GetMediaPropertyAtom (  
    Media      theMedia,  
    QTAtomContainer *propertyAtom );
```

`theMedia`

A reference to the media handler for this operation.

`propertyAtom`

A pointer to a QT atom container. On return, the atom container contains the property atoms for the track associated with the media handler.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You can call `GetMediaPropertyAtom` to retrieve the properties of the track associated with the specified media handler. The contents of the returned QT atom container are defined by the media handler.

Special Considerations

The caller is responsible for disposing of the QT atom container.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Handler Properties” (V–2755)

Related Java Methods

```
quicktime.std.movies.media.Media.getPropertyAtom(),
quicktime.std.movies.AtomContainer.fromMediaProperty()
```

See Also

For the corresponding set function, see `SetMediaPropertyAtom` (III–1604).

GetMediaQuality

Returns a media’s quality level value.

```
short GetMediaQuality (
    Media    theMedia );
```

`theMedia`

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result A short integer whose bits indicate quality constants (see below). More than one of these bits may be set to 1.

GetMediaQuality

Function Result Constants

0x00000001

A pixel depth of 1 bit per pixel.

0x00000010

A pixel depth of 2 bits per pixel.

0x00000100

A pixel depth of 4 bits per pixel.

0x00001000

A pixel depth of 8 bits per pixel.

0x00010000

A pixel depth of 16 bits per pixel.

0x00100000

A pixel depth of 32 bits per pixel.

mediaQualityDraft

The lowest quality level (Bits 6 and 7 = 0).

mediaQualityNormal

An acceptable quality level (Bits 6 and 7 = 1).

mediaQualityBetter

A higher quality level (Bits 6 and 7 = 2).

mediaQualityBest

The highest quality level (Bits 6 and 7 = 3).

Discussion

The Movie Toolbox uses this quality value to influence which track of a movie it selects to play on a given computer. This even applies to sound media. The low-order 6 bits specify pixel depths and the upper 2 bits specify quality levels. If a bit is set to 1, the media can be played at the corresponding depth and quality level.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Alternate Tracks” (V-2738)

Related Java Methods

`quicktime.std.movies.media.Media.getQuality()`

See Also

For the corresponding set function, see `SetMediaQuality` (III-1605).

GetMediaSample

Returns a sample from a movie data file.

```
OSErr GetMediaSample (
    MediaHandle          theMedia,
    long                 dataOut,
    long                 maxSizeToGrow,
    long                 *size,
    TimeValue            time,
    TimeValue            *sampleTime,
    TimeValue            *durationPerSample,
    SampleDescriptionHandle sampleDescriptionH,
    long                 *sampleDescriptionIndex,
    long                 maxNumberOfSamples,
    long                 *numberOfSamples,
    short                *sampleFlags );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II-1120) and `GetTrackMedia` (I-535).

dataOut

A handle. The `GetMediaSample` function returns the sample data into this handle. The function increases the size of this handle, if necessary. You can specify the handle's maximum size with the `maxSizeToGrow` parameter.

maxSizeToGrow

The maximum number of bytes of sample data to be returned. The `GetMediaSample` function does not increase the handle specified by the `dataOut` parameter to a size greater than you specify with this parameter. Set this value to 0 to enforce no limit on the number of bytes to be returned.

size

A pointer to a long integer. The `GetMediaSample` function updates the field referred to by the `size` parameter with the number of bytes of sample data returned in the handle specified by the `dataOut` parameter. Set this parameter to NIL if you are not interested in this information.

time

The starting time of the sample to be retrieved. You must specify this value in the media's time scale.

sampleTime

A pointer to a time value. The `GetMediaSample` function updates this time value to indicate the actual time of the returned sample data. (The returned time may differ from the time you specified with the `time` parameter. This will occur if the

GetMediaSample

time you specified falls in the middle of a sample.) If you are not interested in this information, set this parameter to NIL.

`durationPerSample`

A pointer to a time value. The Movie Toolbox returns the duration of each sample in the media. This time value is expressed in the media's time scale. Set this parameter to 0 if you don't want this information.

`sampleDescriptionH`

A handle to a `SampleDescription` (IV-2414) structure. The `GetMediaSample` function returns the sample description corresponding to the returned sample data. The function resizes this handle as appropriate. If you don't want a `SampleDescription` structure, set this parameter to NIL.

`sampleDescriptionIndex`

A pointer to a long integer. The `GetMediaSample` function returns an index value to the `SampleDescription` (IV-2414) structure that corresponds to the returned sample data. You can retrieve the structure by calling `GetMediaSampleDescription` (I-445) and passing this index in the `descH` parameter. If you don't want this information, set this parameter to NIL.

`maxNumberOfSamples`

The maximum number of samples to be returned. The Movie Toolbox does not return more samples than you specify with this parameter. If you set this parameter to 0, the Movie Toolbox uses a value that is appropriate for the media, and returns that value in the field referenced by the `numberOfSamples` parameter.

`numberOfSamples`

A pointer to a long integer. The `GetMediaSample` function updates the field referred to by this parameter with the number of samples it actually returns. If you don't want this information, set this parameter to NIL.

`sampleFlags`

A pointer to a short integer in which `GetMediaSample` returns flags (see below) that describe the sample. Unused flags are set to 0. If you don't want this information, set this parameter to NIL.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See "Error Codes" (IV-2663).

sampleFlags Constants

`mediaSampleNotSync`

This flag is set to 1 if the sample is not a **sync sample** and to 0 if the sample is a sync sample.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Adding Samples to Media Structures” (V–2752)

Related Java Methods

`quicktime.std.movies.media.Media.getSample()`

See Also

You can add samples to movie data files with `AddMediaSample` (I–27).

GetMediaSampleCount

Determines the number of samples in a media.

```
long GetMediaSampleCount (
    Media    theMedia );
```

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result The number of samples in the media.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Samples” (V–2742)

Related Java Methods

`quicktime.std.movies.media.Media.getSampleCount()`

GetMediaSampleDescription

Retrieves a `SampleDescription` (IV–2414) structure from a media.

```
void GetMediaSampleDescription (
    Media          theMedia,
    long           index,
    SampleDescriptionHandle  descH );
```

GetMediaSampleDescription

`theMedia`

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

`index`

The index of the `SampleDescription` (IV–2414) structure to retrieve. This index corresponds to the structure itself, not to the samples in the media.

`descH`

Specifies a handle that is to receive the `SampleDescription` (IV–2414) structure. The Movie Toolbox correctly resizes this handle for the returned structure. If there is no description for the specified index, the function returns this handle unchanged. Your application must allocate and dispose of this handle.

Discussion

The Movie Toolbox identifies a media’s sample descriptions with an index value, ranging from 1 to the number of sample descriptions in the media. Sample description indexes provide a convenient way to access each sample description in a media. You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Special Considerations

The format of sample descriptions differs by media type. Sample descriptions for image data are defined by `ImageDescription` (IV–2285) structures. Sample descriptions for sound are defined by `SoundDescription` (IV–2449) structures. Sample descriptions for text are defined by `TextDescription` (IV–2474) structures.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Samples” (V–2742)

Related Java Methods

```
quicktime.std.movies.media.Media.getSampleDescription(),
quicktime.std.movies.media.MusicMedia.getMusicDescription(),
quicktime.std.movies.media.VideoMedia.getImageDescription(),
quicktime.std.movies.media.SoundMedia.getSoundDescription(),
quicktime.std.movies.media.SpriteMedia.getSoundDescription(),
quicktime.std.movies.media.ThreeDMedia.getThreeDDescription(),
quicktime.std.movies.media.TextMedia.getTextDescription(),
quicktime.std.movies.media.TimeCodeMedia.getTimeCodeDescription()
```

See Also

For the corresponding set function, see `SetMediaSampleDescription` (III–1606).

GetMediaSampleDescriptionCount

Returns the number of sample descriptions in a media.

```
long GetMediaSampleDescriptionCount (
    Media    theMedia );
```

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result The number of sample descriptions in the media.

Special Considerations

The format of sample descriptions differs by media type. Sample descriptions for image data are defined by `ImageDescription` (IV–2285) structures. Sample descriptions for sound are defined by `SoundDescription` (IV–2449) structures. Sample descriptions for text are defined by `TextDescription` (IV–2474) structures.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Samples” (V–2742)

Related Java Methods

```
quicktime.std.movies.media.Media.getSampleDescriptionCount()
```

GetMediaSampleReference

Obtains reference information about samples that are stored in a movie data file.

```
OSErr GetMediaSampleReference (
    Media          theMedia,
    long           *dataOffset,
    long           *size,
    TimeValue      time,
    TimeValue      *sampleTime,
    TimeValue      *durationPerSample,
    SampleDescriptionHandle sampleDescriptionH,
    long           *sampleDescriptionIndex,
    long           *maxNumberOfSamples,
    long           *numberOfSamples,
    short          *sampleFlags );
```

GetMediaSampleReference

`theMedia`

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

`dataOffset`

A pointer to a long integer. `GetMediaSampleReference` updates the field referred to by this parameter with the offset to the sample data. This parameter is used differently by each media handler. For example, the hierarchical file system (HFS) media handler returns an offset into the file that contains the media data.

`size`

A pointer to a long integer. `GetMediaSampleReference` updates the field referred to by the `size` parameter with the number of bytes of sample data referred to by the reference. Set this parameter to `NIL` if you are not interested in this information.

`time`

The starting time of the sample reference to be retrieved. You must specify this value in the media's time scale.

`sampleTime`

A pointer to a time value. `GetMediaSampleReference` updates this time value to indicate the actual time of the returned sample data. (The returned time may differ from the time you specified with the `time` parameter. This will occur if the time you specified falls in the middle of a sample.) If you are not interested in this information, set this parameter to `NIL`.

`durationPerSample`

A pointer to a time value. The Movie Toolbox returns the duration of each sample in the media. This time value is expressed in the media's time scale. Set this parameter to 0 if you don't want this information.

`sampleDescriptionH`

A handle to a `SampleDescription` (IV–2414) structure. `GetMediaSampleReference` returns the structure corresponding to the returned sample data. The function resizes this handle as appropriate. If you don't want the `SampleDescription` structure, set this parameter to `NIL`.

`sampleDescriptionIndex`

A pointer to a long integer. `GetMediaSampleReference` returns an index value to the `SampleDescription` (IV–2414) structure that corresponds to the returned sample data. To retrieve the media sample description, pass this index in the `descH` parameter of `GetMediaSampleDescription` (I–445). If you don't want this information, set this parameter to `NIL`.

`maxNumberOfSamples`

The maximum number of samples to be returned. The Movie Toolbox does not return a reference that refers to more samples than you specify with this parameter. If you set this parameter to 0, the Movie Toolbox uses a value that is appropriate for the media and returns that value in the field referenced by the `numberOfSamples` parameter.

`numberOfSamples`

A pointer to a long integer. `GetMediaSampleReference` updates the field referred to by this parameter with the number of samples referred to by the returned reference. If you don't want this information, set this parameter to `NIL`.

`sampleFlags`

A pointer to a short integer in which `GetMediaSampleReference` returns flags (see below) that describe the samples referred to by the returned reference. Unused flags are set to 0. If you don't want this information, set this parameter to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

sampleFlags Constants

`mediaSampleNotSync`

This flag is set to 1 if the sample is not a **sync sample** or to 0 if the sample is a sync sample.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Adding Samples to Media Structures” (V-2752)

Related Java Methods

`quicktime.std.movies.media.Media.getSampleReference()`

See Also

See `GetMediaSampleReferences` (I-449).

GetMediaSampleReferences

Obtains reference information about groups of samples that are stored in a movie.

```
OSErr GetMediaSampleReferences (
    Media          theMedia,
    TimeValue      time,
```

GetMediaSampleReferences

TimeValue	*sampleTime,
SampleDescriptionHandle	sampleDescriptionH,
long	*sampleDescriptionIndex,
long	maxNumberOfEntries,
long	*actualNumberOfEntries,
SampleReferencePtr	sampleRefs);

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

time

The starting time of the sample references to be retrieved. You must specify this value in the media’s time scale.

sampleTime

A pointer to a time value. `GetMediaSampleReferences` updates this time value to indicate the actual time of the first returned sample data. If you are not interested in this information, set this parameter to `NIL`.

sampleDescriptionH

A handle to a `SampleDescription` (IV–2414) structure. `GetMediaSampleReference` (I–447) returns the structure corresponding to the returned sample data. The function resizes this handle as appropriate. `GetMediaSampleReferences` only returns a single sample description. If the sample description changes within the media, `GetMediaSampleReferences` returns only as many samples as use a single sample description. You must call it again to get the next group of samples using the next sample description. If you don’t want the `SampleDescription` structure, set this parameter to `NIL`.

sampleDescriptionIndex

A pointer to a long integer. `GetMediaSampleReferences` returns an index value to the `SampleDescription` (IV–2414) structures that correspond to the returned sample data. Use this index to retrieve the media sample descriptions with `GetMediaSampleDescription` (I–445). If you don’t want this information, set this parameter to `NIL`.

maxNumberOfEntries

The maximum number of entries to be returned. The sample references pointer provided by the `sampleRefs` parameter must be large enough to receive the number of entries specified by this parameter. The toolbox does not return more entries than you specify with this parameter. It may, however, return fewer.

`actualNumberOfEntries`

A pointer to a long integer. `GetMediaSampleReferences` updates the field referred to by this parameter with the number of entries referred to by the returned reference.

`sampleRefs`

A pointer to the number of `SampleReferenceRecord` (IV–2416) structures specified in the `maxNumberOfEntries` parameter. On return from this call, the number of sample reference records indicated by the value returned in `actualNumberOfEntries` will be filled in.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Using this function instead of `GetMediaSampleReference` (I–447) can greatly increase the performance of operations that need access to information about each sample in a movie. No information is returned from this call that wasn’t previously available from `GetMediaSampleReference`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Adding Samples to Media Structures” (V–2752)

Related Java Methods

`quicktime.std.movies.media.Media.getSampleReferences()`

See Also

See `GetMediaSampleReference` (I–447) and `GetMediaSampleReferences64` (I–451).

GetMediaSampleReferences64

Provides a 64-bit version of `GetMediaSampleReferences` (I–449).

```
OSErr GetMediaSampleReferences64 (
    Media                theMedia,
    TimeValue            time,
    TimeValue            *sampleTime,
    SampleDescriptionHandle sampleDescriptionH,
    long                 *sampleDescriptionIndex,
    long                 maxNumberOfEntries,
    long                 *actualNumberOfEntries,
```

GetMediaSampleReferences64

SampleReference64Ptr sampleRefs);

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

time

The starting time of the sample references to be retrieved. You must specify this value in the media’s time scale.

sampleTime

A pointer to a time value. `GetMediaSampleReferences64` updates this time value to indicate the actual time of the first returned sample data. If you are not interested in this information, set this parameter to `NIL`.

sampleDescriptionH

A handle to a `SampleDescription` (IV–2414) structure. `GetMediaSampleReference` (I–447) returns the sample description corresponding to the returned sample data. The function resizes this handle as appropriate.

`GetMediaSampleReferences` only returns a single structure. If the sample description changes within the media, `GetMediaSampleReferences` returns only as many samples as use a single sample description. You must call it again to get the next group of samples using the next sample description. If you don’t want the `SampleDescription` structure, set this parameter to `NIL`.

sampleDescriptionIndex

A pointer to a long integer. `GetMediaSampleReferences64` returns an index value to the sample descriptions that correspond to the returned sample data. Use this index to retrieve the media sample descriptions with `GetMediaSampleDescription` (I–445). If you don’t want this information, set this parameter to `NIL`.

maxNumberOfEntries

The maximum number of entries to be returned. The sample references pointer provided by the `sampleRefs` parameter must be large enough to receive the number of entries specified by this parameter. The toolbox does not return more entries than you specify with this parameter. It may, however, return fewer.

actualNumberOfEntries

A pointer to a long integer. `GetMediaSampleReferences64` updates the field referred to by this parameter with the number of entries referred to by the returned reference.

sampleRefs

A pointer to the number of SampleReference64Record (IV–2415) structures specified in the `maxNumberOfEntries` parameter. On return from this call, the number of sample reference records indicated by the value returned in `actualNumberOfEntries` will be filled in.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The only difference between this function and `GetMediaSampleReferences` (I–449) is that the `sampleRefs` parameter points to `SampleReference64Record` structures instead of `SampleReferenceRecord` structures.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

See Also

See `GetMediaSampleReferences` (I–449).

GetMediaShadowSync

Returns the sample number of the shadow **sync sample** associated with a given frame difference sample number.

```
OSErr GetMediaShadowSync (
    Media      theMedia,
    long       frameDiffSampleNum,
    long       *syncSampleNum );
```

`theMedia`

Indicates the media in which the shadow **sync sample** has been established and from which the **shadow sync** number is to be obtained.

`frameDiffSampleNum`

The frame difference sample number associated with the desired shadow **sync sample** number.

GetMediaSyncSampleCount

`syncSampleNum`

A pointer to the sample number of the shadow **sync sample**. If the media does not have a **shadow sync** sample, 0 is returned in the `syncSampleNum` parameter.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V–2722)

Related Java Methods

`quicktime.std.movies.media.Media.getShadowSync()`

See Also

For the corresponding set function, see `SetMediaShadowSync` (III–1607).

GetMediaSyncSampleCount

Gets the number of **sync samples** in a media.

```
long GetMediaSyncSampleCount (
    Media    theMedia );
```

`theMedia`

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result The number of **sync samples** in the media.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.media.Media.getSyncSampleCount()`

GetMediaTimeScale

Determines a media's time scale.

```
TimeScale GetMediaTimeScale (
    Media    theMedia );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result The media's time scale.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Time” (V–2735)

Related Java Methods

`quicktime.std.movies.media.Media.getTimeScale()`

See Also

For the corresponding set function, see `SetMediaTimeScale` (III–1608).

GetMediaTrack

Determines the track that uses a specified media.

```
Track GetMediaTrack (
    Media    theMedia );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result The track identifier of the track that uses the specified media.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Locating a Movie's Tracks and Media Structures” (V–2736)

GetMediaUserData

Related Java Methods

```
quicktime.std.movies.Track.fromMedia(),  
quicktime.std.movies.media.Media.getTrack()
```

GetMediaUserData

Obtains access to a media's **user data list**.

```
UserData GetMediaUserData (  
    Media    theMedia );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

function result The media's **user data list**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

```
quicktime.std.movies.media.UserData.fromMedia(),  
quicktime.std.movies.media.Media.getUserData()
```

See Also

You can then use `GetUserData` (I–547), `AddUserData` (I–44), and `RemoveUserData` (III–1459) to manipulate the contents of the **user data list**. If the data list contains text data, you can use `GetUserDataText` (I–549), `AddUserDataText` (I–45), and `RemoveUserDataText` (III–1460) to work with its contents.

GetMovieActive

Determines whether a movie is currently active.

```
Boolean GetMovieActive (  
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result TRUE if the movie is currently active, FALSE otherwise.

Special Considerations

The Movie Toolbox services only active movies.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Disabling Movies and Tracks” (V–2724)

Related Java Methods

`quicktime.std.movies.Movie.getActive()`

See Also

For the corresponding set function, see `SetMovieActive` (III–1609).

GetMovieActiveSegment

Determines what portion of a movie is currently active for playing.

```
void GetMovieActiveSegment (
    Movie      theMovie,
    TimeValue  *startTime,
    TimeValue  *duration );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

startTime

A pointer to a time value. `GetMovieActiveSegment` places the starting time of the active segment into the field referred to by this parameter. If the returned time value is set to –1, the entire movie is active. In this case, the Movie Toolbox does not return any duration information.

duration

A pointer to a time value. `GetMovieActiveSegment` places the duration of the active movie segment into the field referred to by this parameter. If the entire

GetMovieAnchorDataRef

movie is active, the `startTime` parameter is set to `-1` and this parameter does not return any duration information.

function result You can access this function's error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V-2722)

Related Java Methods

`quicktime.std.movies.Movie.getActiveSegment()`

See Also

For the corresponding set function, see `SetMovieActiveSegment` (III-1609).

GetMovieAnchorDataRef

Retrieves a movie's anchor data reference and type.

```
OSErr GetMovieAnchorDataRef (
    Movie      theMovie,
    Handle     *dataRef,
    OSType     *dataRefType,
    long       *outFlags );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), and `NewMovieFromHandle` (II-1084).

dataRef

A handle to the data reference. The type of information stored in the handle depends upon the data reference type specified by *dataRefType*.

dataRefType

The type of data reference; see “Data References” (IV-2661).

outFlags

If there is no anchor data reference associated with the movie, then `GetMovieAnchorDataRef` sets this parameter to `kMovieAnchorDataRefIsDefault` (see below) and returns copies of the default data reference and type.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

outFlags Constants

`kMovieAnchorDataRefIsDefault`

Data returned in `dataRef` and `dataRefType` are the movie’s default values.

Discussion

If there is neither an anchor nor a default data reference, `NIL` will be returned in `dataRef` and 0 in `dataRefType`.

Special Considerations

The caller should dispose of the data reference returned.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

See Also

For the corresponding set function, see `SetMovieAnchorDataRef` (III–1610).

GetMovieBoundsRgn

Determines a movie’s boundary region.

```
RgnHandle GetMovieBoundsRgn (
    Movie    theMovie );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result A handle to a `MacRegion` (IV–2303) structure that the function allocates. If the movie does not have a spatial representation at the current time, the function returns an empty region. If the function could not satisfy the request, it sets the returned handle to `NIL`.

Discussion

The Movie Toolbox derives the boundary region only from enabled tracks, and only from those tracks that are used in the current display mode (that is, movie or **preview**). The boundary region is valid for the current movie time.

GetMovieBox

Special Considerations

Your application must dispose of the returned region when it is done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

```
quicktime.qd.Region.fromMovieBounds(),  
quicktime.std.movies.Movie.getBoundsRgn()
```

GetMovieBox

Returns a movie’s boundary rectangle, which is a rectangle that encompasses all of the movie’s enabled tracks.

```
void GetMovieBox (  
    Movie    theMovie,  
    Rect     *boxRect );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

boxRect

A pointer to a rectangle. `GetMovieBox` returns the coordinates of the movie’s boundary rectangle into the structure referred to by this parameter.

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Discussion

The movie box is in the coordinate system of the movie’s graphics world and defines the movie’s boundaries over the entire duration of the movie. The movie’s boundary rectangle defines the size and shape of the movie before the Movie Toolbox applies the display clipping region. The following code sample illustrates the use of `GetMovieBox`:

```
// GetMovieBox coding example  
// See “Discovering QuickTime,” page 218  
void main (void)
```

```

{
    WindowPtr      pMacWnd;
    Rect           rectWnd;
    Rect           rectMovie;
    Movie          movie;
    Boolean        bDone = FALSE;
    OSErr          nErr;
    EventRecord     er;
    WindowPtr      pWhichWnd;
    short          nPart;

    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NIL);
    nErr = EnterMovies();
    if (nErr != noErr)
        return;

    SetRect(&rectWnd, 100, 100, 200, 200);
    pMacWnd = NewCWindow(NIL, &rectWnd, "\pMovie", FALSE,
                        noGrowDocProc, (WindowPtr)-1, TRUE, 0);

    SetPort(pMacWnd);
    movie = GetMovie();
    if (movie == NIL)
        return;

    GetMovieBox(movie, &rectMovie);
    OffsetRect(&rectMovie, -rectMovie.left, -rectMovie.top);
    SetMovieBox(movie, &rectMovie);

    SizeWindow(pMacWnd, rectMovie.right, rectMovie.bottom, TRUE);
    ShowWindow(pMacWnd);
    SetMovieGWorld(movie, (CGrafPtr)pMacWnd, NIL);

    StartMovie(movie);

    . . .

```

GetMovieClipRgn

```
        DisposeMovie(movie);
        DisposeWindow(pMacWnd);
    }
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Movie.getBounds()`, `quicktime.std.movies.Movie.getBox()`

See Also

For the corresponding set function, see `SetMovieBox` (III–1611).

GetMovieClipRgn

Determines a movie’s clipping region.

```
RgnHandle GetMovieClipRgn (
    Movie      theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result A handle to a `MacRegion` (IV–2303) structure, which the function allocates, that represents the clipping region. If the function could not satisfy your request or if there is no clipping region defined for the movie, `GetMovieClipRgn` sets the returned handle to `NIL`.

Discussion

The clipping region is saved with the movie when your application saves the movie.

Special Considerations

Your application must dispose of this region when it is done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.qd.Region.fromMovieClip()`, `quicktime.std.movies.Movie.getClipRgn()`

See Also

For the corresponding set function, see `SetMovieClipRgn` (III–1612).

GetMovieColorTable

Retrieves a movie’s color table.

```
OSErr GetMovieColorTable (
    Movie          theMovie,
    CTabHandle     *ctab );
```

theMovie

The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

ctab

A pointer to a field that is to receive a handle to the movie’s color table. If the movie does not have a color table, the toolbox sets the field to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The toolbox returns a copy of the color table, so it is your responsibility to dispose of the color table when you are done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Movie.getColorTable()`,
`quicktime.qd.ColorTable.fromMovie()`

GetMovieCoverProcs

See Also

For the corresponding set function, see `SetMovieColorTable` (III–1613).

GetMovieCoverProcs

Retrieves the cover functions that you set with the `SetMovieCoverProcs` function.

```
OSErr GetMovieCoverProcs (
    Movie          theMovie,
    MovieRgnCoverUPP *uncoverProc,
    MovieRgnCoverUPP *coverProc,
    long           *refcon );
```

theMovie

The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

uncoverProc

Where to return the current uncover procedure. This value is set to `NIL` if no uncover procedure was specified.

coverProc

Where to return the current cover procedure. This value is set to `NIL` if no cover procedure was specified.

refcon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your cover functions need.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

This function returns the uncover and cover functions for the movie as well as the reference constant for the cover functions.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Progress and Cover Functions” (V–2726)

See Also

For the corresponding set function, see `SetMovieCoverProcs` (III–1614).

GetMovieCreationTime

Returns the movie’s creation date and time information.

```
long GetMovieCreationTime (
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result The movie’s creation date and time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Determining Movie Creation and Modification Time” (V–2741)

Related Java Methods

```
quicktime.std.movies.Movie.getCreationTime()
```

GetMovieDataSize

Determines the size of the sample data in a segment of a movie.

```
long GetMovieDataSize (
    Movie    theMovie,
    TimeValue startTime,
    TimeValue duration );
```

theMovie

The movie for this operation. You obtain this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

startTime

A time value specifying the starting point of the segment.

GetMovieDataSize64

duration

A time value that specifies the duration of the segment.

function result The size, in bytes, of the sample data in the defined segment of the designated movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Samples” (V–2742)

Related Java Methods

`quicktime.std.movies.Movie.getDataSize()`

See Also

See `GetMovieDataSize64` (I–466).

GetMovieDataSize64

Provides a 64-bit version of `GetMovieDataSize` (I–465).

```
OSErr GetMovieDataSize64 (
    Movie          theMovie,
    TimeValue      startTime,
    TimeValue      duration,
    wide           *dataSize );
```

theMovie

The movie for this operation. You obtain this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

startTime

A time value specifying the starting point of the segment.

duration

A time value that specifies the duration of the segment.

data size

The size, in bytes, of the sample data in the defined segment of the designated movie.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The only difference between this function and `GetMovieDataSize` (I–465) is that the `dataSize` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

See Also

See `GetMovieDataSize` (I–465).

GetMovieDefaultDataRef

Gets a movie's default data reference.

```
OSErr GetMovieDefaultDataRef (
    Movie      theMovie,
    Handle     *dataRef,
    OSType     *dataRefType );
```

theMovie

A movie identifier. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

dataRef

A pointer to a field that is to receive a handle to the data reference. The function returns a handle to information that identifies the file that contains this media's data. The type of information stored in that handle depends upon the value of the `dataRefType` parameter. If the function cannot locate the specified data reference, the handler sets this returned value to `NIL`. Set the `dataRef` parameter to `NIL` if you are not interested in this information.

dataRefType

A pointer to a field that is to receive the type of data reference; see “Data References” (IV–2661). If the data reference is an **alias**, the function sets this value to `'alis'`, indicating that the reference is an alias. Set the `dataRefType` parameter to `NIL` if you are not interested in this information.

function result You can access Movie Toolbox error returns through `GetMoviesError`

GetMovieDisplayBoundsRgn

(I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

```
quicktime.std.movies.Movie.getDefaultDataRef(),  
quicktime.std.movies.media.DataRef.fromMovie()
```

See Also

For the corresponding set function, see `SetMovieDefaultDataRef` (III–1616).

GetMovieDisplayBoundsRgn

Determines a movie’s display boundary region.

```
RgnHandle GetMovieDisplayBoundsRgn (  
    Movie      theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result A handle to a `MacRegion` (IV–2303) structure that the function allocates. If the movie does not have a spatial representation at the current time, the function returns an empty region. If the function could not satisfy the request, it sets the returned handle to `NIL`.

Discussion

The display boundary region encloses all of a movie’s enabled tracks after the track matrix, track clip, movie matrix, and movie clip have been applied to them. This region is in the display coordinate system of the movie’s graphics world. The Movie Toolbox derives the display boundary region only from enabled tracks, and only from those tracks that are used in the current display mode (that is, movie, **poster**, or **preview**). The display boundary region is valid for the current movie time.

Special Considerations

Your application must dispose of the returned handle when it is done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

```
quicktime.qd.Region.fromMovieDisplayBounds(),  
quicktime.std.movies.Movie.getDisplayBoundsRgn()
```

GetMovieDisplayClipRgn

Determines a movie’s current display clipping region.

```
RgnHandle GetMovieDisplayClipRgn (  
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result A handle to a `MacRegion` (IV–2303) structure that the function allocates. If the movie does not have a spatial representation at the current time, the function returns an empty region. If the function could not satisfy the request, it sets the returned handle to `NIL`.

Special Considerations

Your application must dispose of the returned handle when it is done with it. Note that the display clipping region is not saved with the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

```
quicktime.qd.Region.fromMovieDisplayClip(),  
quicktime.std.movies.Movie.getDisplayClipRgn()
```

See Also

For the corresponding set function, see `SetMovieDisplayClipRgn` (III–1616).

GetMovieDuration

GetMovieDuration

Returns the duration of a movie.

```
TimeValue GetMovieDuration (  
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result The duration of the designated movie.

Discussion

This function returns a time value, expressed in the movie’s time scale, that is calculated to be the maximum durations of all the tracks in the movie. The following code sample illustrates its use:

```
// GetMovieDuration coding example  
// See “Discovering QuickTime,” page 363  
Movie            movie1;  
TimeValue        lOldDuration;  
Movie            movie2;  
long             lIndex, lOrigTrackCount, lReferenceIndex;  
Track            track, trackSprite;  
  
// get the first track in original movie and position at the start  
trackSprite = GetMovieIndTrack(movie1, 1);  
SetMovieSelection(movie1, 0, 0);  
  
// remove all tracks except video in modifier movie  
for (lIndex = 1; lIndex <= GetMovieTrackCount(movie2); lIndex++) {  
    Track        track = GetMovieIndTrack(movie2, lIndex);  
    OSType        dwType;  
  
    GetMediaHandlerDescription(GetTrackMedia(track),  
                              &dwType, NIL, NIL);  
    if (dwType != VideoMediaType) {  
        DisposeMovieTrack(track);  
        lIndex--;  
    }  
}
```

```

// add the modifier track to original movie
10ldDuration = GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);

// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, 10ldDuration,
                    GetMovieDuration(movie1) - 10ldDuration);

// associate the modifier track with the original sprite track
track = GetMovieIndTrack(movie1, 10origTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                  &lReferenceIndex);

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Time” (V-2732)

Related Java Methods

`quicktime.std.movies.Movie.getDuration()`

GetMovieGWorld

Returns a movie’s graphics world.

```

void GetMovieGWorld (
    Movie      theMovie,
    CGrafPtr   *port,
    GDHandle   *gdh );

```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), and `NewMovieFromHandle` (II-1084).

port

A pointer to a field that is to receive a pointer to a `CGrafPort` (IV-2168) structure. Set this parameter to `NIL` if you don’t want this information.

GetMovieImporterForDataRef

gdh

A pointer to a field that is to receive a handle to a GDevice (IV–2264) structure. Set this parameter to NIL if you don’t want this information.

function result You can access this function’s error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

quicktime.std.movies.Movie.getGWorld(), quicktime.qd.QDGraphics.fromMovie()

See Also

For the corresponding set function, see SetMovieGWorld (III–1619).

GetMovieImporterForDataRef

Gets the movie importer component for a movie.

```
OSErr GetMovieImporterForDataRef (
    OSType      dataRefType,
    Handle      dataRef,
    long        flags,
    Component    *importer );
```

dataRefType

The type of data reference; see “Data References” (IV–2661).

dataRef

A handle to the data reference. The type of information stored in the handle depends upon the data reference type specified by *dataRefType*.

flags

Flags (see below) that modify this function’s behavior.

importer

A pointer to an importer component that can import the movie. Returns NIL if no importer can be found.

function result If this function is allowed to use async calls (by being passed `kGetMovieImporterUseAsyncCalls` in the *flags* parameter), it returns

notEnoughDataErr if it would block. You can access this error return through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. For other errors, see “Error Codes” (IV–2663).

flags Constants

kGetMovieImporterDontConsiderGraphicsImporters

Don’t consider any still image file formats that could be opened as QuickTime movies.

kGetMovieImporterUseAsyncCalls

Call DataHGetMIMETypeAsync (I–201) if instructed to do so. Without this flag, the call may block.

Discussion

You can use GetMovieImporterForDataRef to determine if a file can be opened by QuickTime as a movie (for example, in a drag-and-drop operation) as illustrated below:

```
AliasHandle      alias;
MovieImportComponent  mi;

NewAliasMinimal(&reply.sfFile, &alias);
GetMovieImporterForDataRef(rAliasType, (Handle)alias,
kGetMovieImporterDontConsiderGraphicsImporters, &mi);
DisposeHandle((Handle)alias);

if (mi != NIL) {
    // this file can be opened as a movie
    . . .
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Related Java Methods

quicktime.std.qtcomponents.MovieImporter.MovieImporter()

GetMovieIndTrack

Determines the track identifier of a track, given the track’s index value.

GetMovieIndTrack

```
Track GetMovieIndTrack (  
    Movie    theMovie,  
    long     index );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), and `NewMovieFromHandle` (II-1084).

index

The index value of the track for this operation.

function result A track identifier. If the function cannot locate the track, it sets this returned value to `NIL`.

Discussion

This function returns the track identifier that is appropriate to the specified track. The index value identifies the track among all current tracks in a movie. Index values range from 1 to the number of tracks in the movie. The following code sample illustrates its use:

```
// GetMovieIndTrack coding example  
// See "Discovering QuickTime," page 363  
Movie          movie1;  
TimeValue      101dDuration;  
Movie          movie2;  
long           lIndex, lOrigTrackCount, lReferenceIndex;  
Track          track, trackSprite;  
  
// get the first track in original movie and position at the start  
trackSprite = GetMovieIndTrack(movie1, 1);  
SetMovieSelection(movie1, 0, 0);  
  
// remove all tracks except video in modifier movie  
for (lIndex = 1; lIndex <= GetMovieTrackCount(movie2); lIndex++) {  
    Track      track = GetMovieIndTrack(movie2, lIndex);  
    OSType     dwType;  
  
    GetMediaHandlerDescription(GetTrackMedia(track),  
                              &dwType, NIL, NIL);  
    if (dwType != VideoMediaType) {  
        DisposeMovieTrack(track);  
        lIndex--;  
    }  
}
```



```

    }
}

// add the modifier track to original movie
10ldDuration = GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);

// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, 10ldDuration,
    GetMovieDuration(movie1) - 10ldDuration);

// associate the modifier track with the original sprite track
track = GetMovieIndTrack(movie1, 10origTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
    &lReferenceIndex);

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Locating a Movie’s Tracks and Media Structures”
(V–2736)

Related Java Methods

`quicktime.std.movies.Movie.getIndTrack()`

GetMovieIndTrackType

Searches for all of a movie’s tracks that share a given media type or media characteristic.

```

Track GetMovieIndTrackType (
    Movie    theMovie,
    long     index,
    OSType   trackType,
    long     flags );

```

`theMovie`

The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

GetMovieIndTrackType

`index`

The index value of the track for this operation. This is not that same as the track's index value in the movie. Rather, this parameter is an index into the set of tracks that meet your other selection criteria.

`trackType`

Contains either a media type or a media characteristic value. The toolbox applies this value to the search, and returns information about tracks that meet this criterion. You indicate whether you have specified a media type or characteristic value by setting the `flags` parameter appropriately.

`flags`

Contains flags (see below) that control the search operation. Note that you may not set both `movieTrackMediaType` and `movieTrackCharacteristic` to 1.

function result A track identifier.

flags Constants

`movieTrackMediaType`

Indicates that the `trackType` parameter contains a media type value. Set this flag to 1 if you are supplying a media type value (such as `VideoMediaType`).

`movieTrackCharacteristic`

Indicates that the `trackType` parameter contains a media characteristic value. Set this flag to 1 if you are supplying a media characteristic value (such as `VisualMediaCharacteristic`).

`movieTrackEnabledOnly`

Specifies that the toolbox should only search enabled tracks. Set this track to 1 to limit the search to enabled tracks.

Discussion

The toolbox returns the track identifier that corresponds to the track that meets your selection criteria. If the toolbox cannot find a matching track, it returns a value of `NIL`. Note that the `index` parameter does not work the same way that it does in `GetMovieIndTrack` (I-473). With `GetMovieIndTrackType`, the `index` parameter specifies an index into the set of tracks that meet your other selection criteria. For example, in order to find the third track that supports the sound characteristic, you would call the function in the following manner:

```
theTrack = GetMovieIndTrackType (theMovie, 3, AudioMediaCharacteristic,  
movieTrackCharacteristic);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Locating a Movie’s Tracks and Media Structures”
(V–2736)

Related Java Methods

`quicktime.std.movies.Movie.getIndTrackType()`

See Also

See `GetMovieIndTrack` (I–473).

GetMovieLoadState

Returns a value that indicates the state of a movie’s loading process.

```
long GetMovieLoadState (
    Movie    theMovie );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result A constant (see below) that indicates the movie’s loading status.

Function Result Constants

`kMovieLoadStateLoading`

Searching for movie **resource**.

`kMovieLoadStatePlayable`

Movie fully formed; fast-start would work.

`kMovieLoadStatePlaythroughOK`

Returned after the movie has become playable, as soon as QuickTime calculates that the download would be complete before the playback would finish.

`kMovieLoadStateComplete`

All media data is available.

`kMovieLoadStateError`

The loading of the movie failed, typically meaning that a URL could not be found or loaded. This constant has a negative value.

GetMovieMatrix

Discussion

This function lets your code perform relative comparisons against movie loading milestones to determine if certain operations make sense. Its return values are ordered so that they conform to this rule:

```
kMovieLoadStateError  
< kMovieLoadStateLoading  
< kMovieLoadStatePlayable  
< kMovieLoadStateComplete
```

Special Considerations

Because of the “voting system” involved, an application checking for the load state should throttle its calling of the routine. Not calling `GetMovieLoadState` more often than every quarter of a second is a good place to start.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

GetMovieMatrix

Retrieves a movie’s transformation matrix.

```
void GetMovieMatrix (  
    Movie          theMovie,  
    MatrixRecord   *matrix );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

matrix

A pointer to a `MatrixRecord` (IV–2304) structure, where `GetMovieMatrix` returns the movie’s matrix.

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Movie.getMatrix()`

See Also

For the corresponding set function, see `SetMovieMatrix` (III–1622). You can set a movie’s matrix with `SetMovieMatrix`. The Movie Toolbox provides a number of functions that allow you to manipulate movie matrices. See `SetIdentityMatrix` (III–1593) for information about these functions.

GetMovieModificationTime

Returns a movie’s modification date and time.

```
long GetMovieModificationTime (  
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result The movie’s modification date and time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Determining Movie Creation and Modification Time” (V–2741)

Related Java Methods

`quicktime.std.movies.Movie.getModificationTime()`

GetMovieNaturalBoundsRect

Gets a movie’s natural boundary rectangle.

```
void GetMovieNaturalBoundsRect (  
    Movie    theMovie,
```

GetMovieNextInterestingTime

```
Rect          *naturalBounds );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

naturalBounds

A pointer to a `Rect` (IV–2399) structure that represents the movie’s bounding rectangle.

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.Movie.getNaturalBoundsRect()`

GetMovieNextInterestingTime

Searches for times of interest in a movie’s enabled tracks.

```
void GetMovieNextInterestingTime (
    Movie          theMovie,
    short          interestingTimeFlags,
    short          numMediaTypes,
    const OSType   *whichMediaTypes,
    TimeValue      time,
    Fixed          rate,
    TimeValue      *interestingTime,
    TimeValue      *interestingDuration );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

`interestingTimeFlags`

Contains flags (see below) that determine the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit`, `nextTimeTrackEdit` and `nextTimeSyncSample` flags to 1. Set unused flags to 0.

`numMediaTypes`

The number of media types in the table referred to by the `whichMediaType` parameter. Set this parameter to 0 to search all media types.

`whichMediaTypes`

A pointer to an array of media type constants (see below). You can use this parameter to limit the search to a specified set of media types. Each entry in the table referred to by this parameter identifies a media type to be included in the search. You use the `numMediaTypes` parameter to indicate the number of entries in the table. Set this parameter to `NIL` to search all media types.

`time`

Specifies a time value that establishes the starting point for the search. This time value must be expressed in the movie's time scale.

`rate`

The search direction. Negative values cause the Movie Toolbox to search backward from the starting point specified in the `time` parameter. Other values cause a forward search.

`interestingTime`

A pointer to a time value. The Movie Toolbox returns the first time value it finds that meets the search criteria specified in the `flags` parameter. This time value is in the movie's time scale. If there are no times that meet the search criteria you specify, the Movie Toolbox sets this value to `-1`. If you are not interested in this information, set this parameter to `NIL`.

`interestingDuration`

A pointer to a time value. The Movie Toolbox returns the duration of the interesting time. This time value is in the movie's time coordinate system. Set this parameter to `NIL` if you don't want this information; in this case, the function works faster.

interestingTimeFlags Constants

`nextTimeMediaSample`

Searches for the next sample in the movie's media. Set this flag to 1 to search for the next sample.

`nextTimeMediaEdit`

Searches for the next group of samples in the movie's media. Set this flag to 1 to search for the next group of samples.

GetMovieNextInterestingTime

`nextTimeTrackEdit`

Searches for the media sample that corresponds to the next entry in a track's media edit list. The end of the track is considered an empty edit. Set this flag to 1 to search for the next track edit.

`nextTimeSyncSample`

Searches for the next **sync sample** in the movie's media. Set this flag to 1 to search for the next sync sample. Sync samples don't rely on preceding frames for content. Some compression algorithms conserve space by eliminating duplication between consecutive frames in a sample.

`nextTimeEdgeOK`

Instructs the Movie Toolbox that you are willing to receive information about elements that begin or end at the time specified by the `time` parameter. Set this flag to 1 to accept this information. When this flag is set, the function returns valid information about the beginning and end of the movie.

`nextTimeIgnoreActiveSegment`

Instructs the Movie Toolbox to look outside of the active segment for samples that meet the search criteria. Set this flag to 1 to search outside of the active segment.

whichMediaTypes Constants

`VisualMediaCharacteristic`

Value = 'eyes'. Instructs the Movie Toolbox to search all tracks that have spatial bounds.

`AudioMediaCharacteristic`

Value = 'ears'. Instructs the Movie Toolbox to search all tracks that play sound.

Discussion

The following code sample shows the use of `GetMovieNextInterestingTime` to return, through the `time` parameter, the starting time of the first video sample of the specified QuickTime movie. The trick here is to set the `nextTimeEdgeOK` flag, to indicate that you want to get the starting time of the beginning of the movie. If this function encounters an error, it returns a (bogus) starting time of -1, as shown below:

```
static OSErr QTStep_GetStartTimeOfFirstVideoSample (Movie theMovie,
                                                    TimeValue *theTime)
{
    short          myFlags;
    OSType         myTypes[1];

    *theTime = kBogusStartingTime;           // a bogus starting time
```



```

    if (theMovie == NIL)
        return(invalidMovie);

    myFlags = nextTimeMediaSample + nextTimeEdgeOK;
                // we want the first sample in the movie
    myTypes[0] = VisualMediaCharacteristic;    // we want video samples

    GetMovieNextInterestingTime(theMovie, myFlags, 1, myTypes,
                                (TimeValue)0, fixed1, theTime, NIL);

    return(GetMoviesError());
}

```

Special Considerations

This function examines only the movie's enabled tracks.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Finding Interesting Times" (V-2735)

Related Java Methods

`quicktime.std.movies.Movie.getNextInterestingTime()`

GetMoviePict

Creates a QuickDraw picture from a specified movie at a specified time.

```

PicHandle GetMoviePict (
    Movie          theMovie,
    TimeValue      time );

```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), and `NewMovieFromHandle` (II-1084).

time

The movie time from which the image is to be taken.

function result A handle to a `Picture` (IV-2331) structure. If the function could not create the picture, the returned handle is set to `NIL`.

GetMoviePosterPict

Discussion

This function uses only those movie tracks that are currently enabled and would therefore be used in playback. Your application may call this function even if the movie is inactive.

Special Considerations

Your application must dispose of this picture handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Generating Pictures From Movies” (V–2725)

Related Java Methods

`quicktime.qd.Pict.fromMovie()`, `quicktime.std.movies.Movie.getPict()`

See Also

See `GetTrackPict` (I–540).

GetMoviePosterPict

Creates a QuickDraw picture that contains a movie’s **poster**.

```
PicHandle GetMoviePosterPict (  
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result A handle to a `Picture` (IV–2331) structure. If the function could not create the picture, the returned handle is set to `NIL`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Generating Pictures From Movies” (V–2725)

Related Java Methods

`quicktime.qd.Pict.fromMovie()`, `quicktime.std.movies.Movie.getPosterPict()`

GetMoviePosterTime

Returns the **poster**'s time in a movie.

```
TimeValue GetMoviePosterTime (
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result The time in the movie from which its **poster** is taken.

Discussion

Since a movie **poster** has no duration, it is defined by a point in time within the movie. The time value returned by `GetMoviePosterTime` is in the time coordinate system of the movie and represents the starting time for the movie frame that contains the poster image.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V–2718)

Related Java Methods

`quicktime.std.movies.Movie.getPosterTime()`

See Also

For the corresponding set function, see `SetMoviePosterTime` (III–1625).

GetMoviePreferredRate

Returns a movie's default playback rate.

```
Fixed GetMoviePreferredRate (
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

GetMoviePreferredVolume

function result The movie's default playback rate.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Preferred Movie Settings" (V-2721)

Related Java Methods

`quicktime.std.movies.Movie.getPreferredRate()`

See Also

For the corresponding set function, see `SetMoviePreferredRate` (III-1626).

GetMoviePreferredVolume

Returns a movie's preferred volume setting.

```
short GetMoviePreferredVolume (  
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), and `NewMovieFromHandle` (II-1084).

function result The movie's preferred volume setting.

Discussion

A movie's tracks have their own volume settings. A track's volume is scaled by the movie's volume to produce the track's final volume. On Macintosh computers, the movie's volume is further scaled by the sound volume that the user controls from the Sound control panel.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Preferred Movie Settings" (V-2721)

Related Java Methods

`quicktime.std.movies.Movie.getPreferredVolume()`

See Also

For the corresponding set function, see `SetMoviePreferredVolume` (III–1627). Use the `SetTrackVolume` (III–1669) function to set the volume of an individual track.

GetMoviePreviewMode

Determines whether a movie is in **preview** mode.

```
Boolean GetMoviePreviewMode (
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result TRUE if the movie is in **preview** mode; FALSE if the movie is in normal playback mode.

Discussion

If a movie is in **preview** mode, only the movie’s preview can be displayed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V–2718)

Related Java Methods

`quicktime.std.movies.Movie.getPreviewMode()`

See Also

For the corresponding set function, see `SetMoviePreviewMode` (III–1628).

GetMoviePreviewTime

Returns the starting time and duration of the movie’s **preview**.

```
void GetMoviePreviewTime (
    Movie    theMovie,
    TimeValue *previewTime,
    TimeValue *previewDuration );
```

GetMovieProgressProc

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

previewTime

A pointer to a time value. The Movie Toolbox places the **preview**’s starting time into the field referred to by this parameter. If the movie does not have a preview, the Movie Toolbox sets this returned value to 0.

previewDuration

A pointer to a time value. The Movie Toolbox places the **preview**’s duration into the field referred to by this parameter. If the movie does not have a preview, the Movie Toolbox sets this returned value to 0.

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V–2718)

Related Java Methods

`quicktime.std.movies.Movie.getPreviewTime()`

See Also

For the corresponding set function, see `SetMoviePreviewTime` (III–1629).

GetMovieProgressProc

Gets the `MovieProgressProc` (III–2105) callback attached to a movie.

```
void GetMovieProgressProc (
    Movie          theMovie,
    MovieProgressUPP *p,
    long           *refcon );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

p

On return, a pointer to a `MovieProgressProc` (III–2105) callback.

refcon

On return, a reference constant passed to the callback. This parameter is used to point to a data structure containing any information the function needs.

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

See Also

For the corresponding set function, see `SetMovieProgressProc` (III–1630).

GetMoviePropertyAtom

Gets a movie’s property atom.

```
OSErr GetMoviePropertyAtom (
    Movie          theMovie,
    QTAtomContainer *propertyAtom );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

propertyAtom

A pointer to a property atom.

function result You can access `Movie Toolbox` error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

This routine is used to author event handlers for the `kQTEventMovieLoaded` QuickTime event.

Version Notes

Introduced in QuickTime 4.1.

GetMovieRate

Programming Info

C interface file: `Movies.h`

See Also

For the corresponding set function, see `SetMoviePropertyAtom` (III–1631).

GetMovieRate

Returns a movie’s playback rate.

```
Fixed GetMovieRate (
    Movie    theMovie );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result The movie’s playback rate.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Time” (V–2732)

Related Java Methods

`quicktime.std.movies.Movie.getRate()`

See Also

For the corresponding set function, see `SetMovieRate` (III–1631).

GetMovieSegmentDisplayBoundsRgn

Determines a movie’s display boundary region for a specified segment.

```
RgnHandle GetMovieSegmentDisplayBoundsRgn (
    Movie        theMovie,
    TimeValue    time,
    TimeValue    duration );
```


theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

time

The starting time of the movie segment to consider. This time value must be expressed in the movie’s time coordinate system. The duration parameter specifies the length of the segment.

duration

The length of the segment to consider. Set this parameter to 0 to specify an instant in time.

function result

A handle to a `MacRegion` (IV–2303) structure that the function allocates. This region is defined in the movie’s display coordinate system. If the movie does not have a spatial representation at the current time, the function returns an empty region. If the function could not satisfy the request, it sets the returned handle to `NIL`.

Discussion

This function allocates a region and returns a handle to it. The Movie Toolbox derives the display boundary region only from enabled tracks and only from those tracks that are used in the current display mode (movie, **poster**, or **preview**). The display boundary region encloses all of a movie’s enabled tracks after the track matrix, track clip, movie matrix, and movie clip have been applied to them.

Special Considerations

Your application must dispose of the returned region when it is done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.qd.Region.fromMovieSegment()`,

`quicktime.std.movies.Movie.getSegmentDisplayBoundsRgn()`

GetMovieSelection

Returns information about a movie’s current selection.

GetMoviesError

```
void GetMovieSelection (
    Movie      theMovie,
    TimeValue  *selectionTime,
    TimeValue  *selectionDuration );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

selectionTime

A pointer to a time value. The `GetMovieSelection` function places the starting time of the current selection into the field referred to by this parameter. Set this parameter to `NIL` if you don't want this information.

selectionDuration

A pointer to a time value. The `GetMovieSelection` function places the duration of the current selection into the field referred to by this parameter. Set this parameter to `NIL` if you don't want this information.

function result You can access this function's error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Movie Editing Functions” (V–2746)

Related Java Methods

`quicktime.std.movies.Movie.getSelection()`

See Also

For the corresponding set function, see `SetMovieSelection` (III–1632).

GetMoviesError

Returns the contents of the current error value and resets the current error value to 0.

```
OSErr GetMoviesError ( void );
```

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error in the current error value.

Discussion

The Movie Toolbox provides two error values to your application: the current error and the **sticky error**. The current error is the result code from the last Movie Toolbox function; it is updated each time your application calls a Movie Toolbox function. The following code sample shows a typical use:

```
// GetMoviesError coding example
// See "Discovering QuickTime," page 256
OSErr QTUtils_SaveMovie (Movie theMovie)
{
    StandardFileReply    mySFReply;
    StringPtr    myPrompt = QTUtils_ConvertCToPascalString(kSavePrompt);
    StringPtr    myFileName =
        QTUtils_ConvertCToPascalString(kSaveMovieFileName);
    OSErr        myErr = noErr;
    if (theMovie == NIL)
        return(invalidMovie);
    StandardPutFile(myPrompt, myFileName, &mySFReply);
    if (mySFReply.sfGood) {
        FlattenMovieData(    theMovie,
                            flattenAddMovieToDataFork,
                            &mySFReply.sfFile,
                            FOUR_CHAR_CODE('TVOD'),
                            smSystemScript,
                            createMovieFileDeleteCurFile);
        myErr = GetMoviesError();
    }
    free(myPrompt);
    free(myFileName);
    return(myErr);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Error Functions" (V-2712)

See Also

See `GetMoviesStickyError` (I-494).

GetMoviesStickyError

GetMoviesStickyError

Returns the contents of the **sticky error** value.

```
OSErr GetMoviesStickyError ( void );
```

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error in the **sticky error** value.

Discussion

The **sticky error** value contains the first nonzero result code from any Movie Toolbox function that you called after having cleared the sticky error with `ClearMoviesStickyError` (I–90).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Error Functions” (V–2712)

See Also

See `GetMoviesError` (I–492).

GetMovieStatus

Searches for errors in all the enabled tracks of the movie and returns information about errors that are encountered during the processing associated with the `MoviesTask` function.

```
ComponentResult GetMovieStatus (
    Movie    theMovie,
    Track    *firstProblemTrack );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

firstProblemTrack

A pointer to a track identifier. The Movie Toolbox places the identifier for the first track that is found to contain an error into the field referred to by this parameter. If you don’t want to receive the track identifier, set this parameter to `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error in the movie status value.

Discussion

This function returns information about errors that are encountered during `MoviesTask` (II–973) execution. These errors typically reflect playback problems, such as low-memory conditions. `GetMovieStatus` returns the error associated with the first problem track.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movies and Your Event Loop” (V–2720)

Related Java Methods

`quicktime.std.movies.Movie.getStatus()`

GetMovieTime

Returns a movie’s current time both as a time value and in a **time structure**.

```
TimeValue GetMovieTime (
    Movie          theMovie,
    TimeRecord     *currentTime );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

currentTime

A pointer to a `TimeRecord` (IV–2486) structure. The function updates this **time structure** to contain the movie’s current time. If you don’t want this information, set this parameter to `NIL`.

function result The time value of the current time.

Discussion

This function returns the movie’s current time value in two formats: as a time value and in a **time structure**.

Version Notes

Introduced in QuickTime 3 or earlier.

GetMovieTimeBase

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Time” (V–2732)

Related Java Methods

`quicktime.std.movies.Movie.getTime()`, `quicktime.std.movies.Movie.getTRTime()`

See Also

For the corresponding set function, see `SetMovieTime` (III–1634).

GetMovieTimeBase

Returns a movie’s time base.

```
TimeBase GetMovieTimeBase (  
    Movie    theMovie );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result The movie’s `TimeBaseRecord` (IV–2481) structure.

Special Considerations

The Movie Toolbox disposes of a movie’s time base when you dispose of the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Time” (V–2732)

Related Java Methods

`quicktime.std.movies.Movie.getTimeBase()`,
`quicktime.std.clocks.TimeBase.fromMovie()`

GetMovieTimeScale

Returns the time scale of a movie.

```
TimeScale GetMovieTimeScale (  
    Movie    theMovie );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result A long integer that contains the movie’s time scale.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Time” (V–2732)

Related Java Methods

`quicktime.std.movies.Movie.getTimeScale()`

See Also

For the corresponding set function, see `SetMovieTimeScale` (III–1635).

GetMovieTrack

Determines the track identifier of a track, given the track’s ID value.

```
Track GetMovieTrack (
    Movie    theMovie,
    long     trackID );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

`trackID`

The ID value of the track for this operation.

function result A track identifier.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Locating a Movie’s Tracks and Media Structures” (V–2736)

GetMovieTrackCount

Related Java Methods

```
quicktime.std.movies.Movie.getTrack()
```

GetMovieTrackCount

Returns the number of tracks in a movie.

```
long GetMovieTrackCount (
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result The number of tracks in the movie.

Discussion

The following code sample illustrates the use of `GetMovieTrackCount`:

```
// GetMovieTrackCount coding example
// See "Discovering QuickTime," page 363
Movie          movie1;
TimeValue      101dDuration;
Movie          movie2;
long           1Index, 1OrigTrackCount, 1ReferenceIndex;
Track          track, trackSprite;

// get the first track in original movie and position at the start
trackSprite = GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);

// remove all tracks except video in modifier movie
for (1Index = 1; 1Index <= GetMovieTrackCount(movie2); 1Index++) {
    Track          track = GetMovieIndTrack(movie2, 1Index);
    OSType          dwType;

    GetMediaHandlerDescription(GetTrackMedia(track),
                              &dwType, NIL, NIL);
    if (dwType != VideoMediaType) {
        DisposeMovieTrack(track);
        1Index--;
    }
}
```



```

    }
}

// add the modifier track to original movie
10ldDuration = GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);

// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, 10ldDuration,
    GetMovieDuration(movie1) - 10ldDuration);

// associate the modifier track with the original sprite track
track = GetMovieIndTrack(movie1, 10origTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
    &lReferenceIndex);

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Locating a Movie’s Tracks and Media Structures”
(V–2736)

Related Java Methods

`quicktime.std.movies.Movie.getTrackCount()`

GetMovieUserData

Obtains access to a movie’s **user data list**.

```

UserData GetMovieUserData (
    Movie    theMovie );

```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result The `UserDataRecord` (IV–2496) structure for the movie. If the function could not locate the movie’s user data, it sets this return value to `NIL`.

GetMovieVolume

Discussion

This function returns a reference to the movie's **user data list**, which is valid until you dispose of the movie. When you save the movie, the Movie Toolbox saves the user data as well.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

```
quicktime.std.movies.media.UserData.fromMovie(),  
quicktime.std.movies.Movie.getUserData()
```

See Also

You can use `GetUserData` (I–547), `AddUserData` (I–44) and `RemoveUserData` (III–1459) to manipulate the contents of the **user data list**. If the data list contains text data, you can use `GetUserDataText` (I–549), `AddUserDataText` (I–45), and `RemoveUserDataText` (III–1460) to work with its contents.

GetMovieVolume

Returns a movie's current volume setting.

```
short GetMovieVolume (  
    Movie    theMovie );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result The current volume setting for the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Sound Volume” (V–2732)

Related Java Methods

```
quicktime.std.movies.Movie.getVolume()
```

See Also

For the corresponding set function, see `SetMovieVolume` (III–1637).

GetNextCallback

Returns the next **callback event** associated with a specified time base.

```
QTCallback GetNextCallback (
    QTCallback    cb );
```

cb

Specifies the starting **callback event** for the operation. Your clock component obtains this value from the `GetFirstCallback` (I–411) function or from previous calls to the `GetNextCallback` function.

function result A pointer to a `CallbackRecord` (IV–2165) structure. Your software can pass this structure to other functions, such as `ClockRateChanged` (I–97).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Toolbox Clock Support Functions” (V–2869)

GetNextImageDescriptionExtensionType

Retrieves an image description structure extension type.

```
OSErr GetNextImageDescriptionExtensionType (
    ImageDescriptionHandle desc,
    long *idType );
```

desc

A handle to an `ImageDescription` (IV–2285) structure.

idType

A pointer to an integer that indicates the type of the extension after which this function is to return the next extension type. Point to a value of 0 to return the first type found.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

GetNextTrackForCompositing

Discussion

This function allows your application to search for all the types of extensions in an `ImageDescription` (IV-2285) structure. The `idType` parameter should be set to 0 to start the search. When no more extension types can be found, the function will set this field to 0.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Image Descriptions” (V-2790)

GetNextTrackForCompositing

Determines the next track in a movie’s compositing process.

```
Track GetNextTrackForCompositing (
    Movie    theMovie,
    Track    theTrack );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), and `NewMovieFromHandle` (II-1084).

theTrack

The identifier of the track from which to start.

function result The returned identifier of the next track to be composited.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.Movie.getNextTrackForCompositing()`

See Also

See `GetPrevTrackForCompositing` (I-506).

GetNextTrackReferenceType

Determines all of the track reference types that are defined for a given track.

```
OSType GetNextTrackReferenceType (
    Track    theTrack,
    OSType   refType );
```

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

refType

The type of reference. Set this parameter to 0 to retrieve the first track reference type. On subsequent requests, use the previous value returned by this function.

function result An OSType containing the next track reference type value defined for the track; see “Data References” (IV–2661).

Discussion

There is no implied ordering of the values returned by this function . When you reach the end of the track’s reference types, this function sets the returned value to 0.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Track References” (V–2737)

Related Java Methods

`quicktime.std.movies.Track.getNextReferenceType()`

GetNextUserDataTypes

Retrieves the next user data type in a specified **user data list**.

```
long GetNextUserDataTypes (
    UserData    theUserData,
    OSType      udType );
```

GetPictureFileHeader

theUserData

The **user data list** for this operation. You obtain this list reference by calling the `GetMovieUserData` (I–499), `GetTrackUserData` (I–546), or `GetMediaUserData` (I–456) function.

udType

Specifies a user data field; see “User Data Identifiers” (IV–2696). Set this parameter to 0 to retrieve the first user data field in the **user data list**. On subsequent requests, use the previous value returned by this function.

function result The next user data type in the list. Returns 0 when there are no more user data types.

Discussion

Use this function to scan all the user data types in a **user data list**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

`quicktime.std.movies.media.UserData.getNextType()`

GetPictureFileHeader

Extracts the picture frame and file header from a specified **picture file**.

```
OSErr GetPictureFileHeader (
    short      refNum,
    Rect       *frame,
    OpenCPicParams *header );
```

refNum

A file reference number for the source PICT file.

frame

A pointer to a rectangle that is to receive the picture frame rectangle of the **picture file**. This function places the `picFrame` rectangle from the picture structure into the rectangle referred to by the `frame` parameter. If you are not interested in this information, pass `NIL` in this parameter.

header

A pointer to an `OpenCPicParams` (IV–2326) structure. The `GetPictureFileHeader` function places the header from the specified **picture file** into this structure. If you are not interested in this information, pass `NIL` in this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your program can use the information returned in the header parameter to determine how to draw an image without having to read the **picture file**.

Special Considerations

Note that this function always returns a version 2 header. If the source file is a version 1 PICT file, the `GetPictureFileHeader` function converts the header into version 2 format before returning it to your application. See *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**) for more information about picture headers.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pictures and PICT Files” (V–2790)

GetPosterBox

Obtains a **poster**’s boundary rectangle.

```
void GetPosterBox (
    Movie    theMovie,
    Rect     *boxRect );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

boxRect

A pointer to a rectangle. The Movie Toolbox returns the **poster**’s boundary rectangle into the structure referred to by this parameter.

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

GetPrevTrackForCompositing

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V–2718)

Related Java Methods

`quicktime.std.movies.Movie.getPosterBox()`

See Also

For the corresponding set function, see `SetPosterBox` (III–1638).

GetPrevTrackForCompositing

Determines the previous track in a movie’s compositing process.

```
Track GetPrevTrackForCompositing (
    Movie    theMovie,
    Track    theTrack );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

theTrack

The identifier of the track from which to start.

function result The returned identifier of the previous track in the compositing process.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.Movie.getPrevTrackForCompositing()`

See Also

See `GetNextTrackForCompositing` (I–502).

GetQuickTimePreference

Retrieves a particular preference from the QuickTime preferences.

```
OSErr GetQuickTimePreference (
    OSType          preferenceType,
    QTAtomContainer *preferenceAtom );
```

preferenceType

A preference type to be retrieved (see below); see “Atom ID Codes” (IV–2649).

preferenceAtom

A pointer to the returned preference atom.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

preferenceType Constants

ConnectionSpeedPrefsType

The connection speed currently set in the QuickTime Preferences; the atom type returned in `preferenceAtom` is 'cspd'.

BandwidthManagementPrefsType

The user’s current **bandwidth** management preference; the atom type returned in `preferenceAtom` is 'bwmg'.

Discussion

The following sample code shows how to retrieve the connection speed setting from the QuickTime preferences:

```
struct ConnectionSpeedPrefsRecord {
    long connectionSpeed;
};
typedef struct ConnectionSpeedPrefsRecord ConnectionSpeedPrefsRecord;
. . .

OSErr          err;
QTAtomContainer prefs;
QTAtom         prefsAtom;
long           dataSize;
Ptr            atomData;
ConnectionSpeedPrefsRecord prefrec;

err = GetQuickTimePreference(ConnectionSpeedPrefsType, &prefs);
```

GetSimilarity

```
if (err == noErr) {
    prefsAtom = QTFindChildByID(prefs, kParentAtomIsContainer,
                                ConnectionSpeedPrefsType, 1, nil);

    if (!prefsAtom) {
        // set the default setting to 28.8kbps
        prefrec.connectionSpeed = kDataRate288ModemRate;
    } else {
        err = QTGetAtomDataPtr(prefs, prefsAtom, &dataSize,
                                &atomData);

        if (dataSize != sizeof(ConnectionSpeedPrefsRecord)) {
            // the prefs record wasn't the right size,
            // so it must be corrupt -- set to the default
            prefrec.connectionSpeed = kDataRate288ModemRate;
        } else {
            // everything was fine -- read the connection speed
            prefrec = *(ConnectionSpeedPrefsRecord *)atomData;
        }
    }
    QTDisposeAtomContainer(prefs);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

See Also

For the corresponding set function, see `SetQuickTimePreference` (III–1639).

GetSimilarity

Compares a compressed image to a picture stored in a pixel map and returns a value indicating the relative similarity of the two images.

```
OSErr GetSimilarity (
    PixMapHandle      src,
    const Rect        *srcRect,
    ImageDescriptionHandle desc,
    Ptr               data,
    Fixed              *similarity );
```

`src`

A handle to the noncompressed image. The image must be stored in a pixel map structure.

`srcRect`

A pointer to a rectangle defining the portion of the image to compare to the compressed image. This rectangle should be the same size as the image described by the `ImageDescription` (IV–2285) structure specified by the `desc` parameter.

`desc`

A handle to the `ImageDescription` (IV–2285) structure that defines the compressed image for the operation.

`data`

Points to the compressed image data. This pointer must contain a **32-bit clean** address.

`similarity`

A pointer to a field that is to receive the similarity value. The compressor sets this field to reflect the relative similarity of the two images. Valid values range from 0 (completely different) to 1.0 (identical).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting Information About Compressed Data” (V–2787)

Related Java Methods

`quicktime.std.image.QTImage.getSimilarity()`

GetSoundDescriptionExtension

Gets the current extension to a `SoundDescription` (IV–2449) structure.

```
OSErr GetSoundDescriptionExtension (
    SoundDescriptionHandle desc,
    Handle *extension,
    OSType idType );
```

`desc`

A handle to a `SoundDescription` (IV–2449) structure.

GetSoundHeaderOffset

extension

A pointer to a handle that, on return, contains the extension.

idType

A four-byte signature that identifies the type of data in the extension.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

```
quicktime.util.QTHandle.fromSoundDescription(),  
quicktime.std.movies.media.SoundDescription.getExtension()
```

GetSoundHeaderOffset

Gets the offset from the beginning of a sound **resource** to the embedded sound header.

```
OSErr GetSoundHeaderOffset (  
    SndListHandle    sndHandle,  
    long             *offset );
```

sndHandle

A handle to a `SndListResource` (IV–2447) structure.

offset

On exit, the offset in bytes from the beginning of the sound **resource** specified by the `sndHdl` parameter to the beginning of the sound header within that sound resource.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns the number of bytes from the beginning of the sound **resource** to the sound header that is contained within that resource. You might need this information if you want to use the address of that sound header in a sound command (such as `soundCmd` or `bufferCmd`). The handle passed to `GetSoundHeaderOffset` does not have to be locked.

Special Considerations

You can call the `GetSoundHeaderOffset` function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

Related Java Methods

`quicktime.sound.SndHandle.getSoundHeaderOffset()`

GetSoundOutputInfo

Gets information about the current sound environment.

```
OSErr GetSoundOutputInfo (
    Component    outputDevice,
    OSType       selector,
    void         *infoPtr );
```

outputDevice

The identifier of a sound output component. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131). To access the default output device, pass `NIL`.

selector

The selector for the information you want; see “Sound Information Selectors” (IV–2693).

infoPtr

A pointer to the returned sound device information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This routine operates directly on the sound output device and can be used to retrieve information about the hardware settings.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding set function, see `SetSoundOutputInfo` (III–1640).

GetSoundPreference

GetSoundPreference

Reads a block of preferences data from a sound **resource** file.

```
OSErr GetSoundPreference (
    OSType    theType,
    Str255    name,
    Handle     settings );
```

theType

The **resource** type of the preferences resource.

name

The **resource** name of the preferences resource.

settings

A handle to the data in the preferences **resource**.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function retrieves the block of preferences data you previously stored in a **resource** by calling `SetSoundPreference` (III–1641). It is the responsibility of your component to allocate the block of data referenced by the `settings` parameter. The Sound Manager resizes the handle (if necessary) and fills it with data from the resource with the specified type and name. Your sound component should dispose of the handle once it has finished reading the data from it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding set function, see `SetSoundPreference` (III–1641).

GetSpriteProperty

Retrieves the value of a specified **sprite** property.

```
OSErr GetSpriteProperty (
    Sprite    theSprite,
    long      propertyType,
    void      *propertyValue );
```

theSprite

The **sprite** for this operation.

propertyType

The property whose value should be retrieved (see below).

propertyValue

A pointer to a variable that will hold the selected property value on return.

Depending on the property type, this parameter is either a pointer to the property value or the property value itself, cast as a void pointer.

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

propertyType Constants

kSpritePropertyMatrix

The propertyValue parameter returns the **sprite**’s MatrixRecord (IV-2304) structure.

kSpritePropertyImageDescription

The propertyValue parameter returns an ImageDescriptionHandle, which is a handle to the **sprite**’s ImageDescription (IV-2285) structure.

kSpritePropertyImageDataPtr

The propertyValue parameter returns a Ptr, which is a pointer to the **sprite**’s image data.

kSpritePropertyVisible

The propertyValue parameter returns a Boolean that is TRUE if the **sprite** is visible, FALSE otherwise.

kSpritePropertyLayer

The propertyValue parameter returns the **sprite**’s layer number.

kSpritePropertyGraphicsMode

The propertyValue parameter returns the **sprite**’s ModifierTrackGraphicsModeRecord (IV-2311).

Discussion

You call this function to retrieve the value of a **sprite** property, setting the propertyType parameter to the type of the property you want to retrieve.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Creating and Manipulating Sprites” (V-2901)

GetSysBeepVolume

Related Java Methods

```
quicktime.util.RawEncodedImage.fromSprite(),
quicktime.std.anim.Sprite.getMatrix(),
quicktime.std.anim.Sprite.getImageDescription(),
quicktime.std.anim.Sprite.getImageData(),
quicktime.std.anim.Sprite.getVisible(),
quicktime.std.anim.Sprite.getLayer(),
quicktime.std.anim.Sprite.getGraphicsMode(),
quicktime.std.anim.Sprite.getImageDataSize(),
quicktime.std.image.ImageDescription.fromSprite()
```

See Also

For the corresponding set function, see `SetSpriteProperty` (III–1641).

GetSysBeepVolume

Determines the current volume of the system alert sound.

```
OSErr GetSysBeepVolume (
    long    *level );
```

level

On exit, a pointer to the current volume level of the system alert sound. The value may range from 0x0000 (silence) to 0x0100 (full volume).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding set function, see `SetSysBeepVolume` (III–1647).

GetTimeBaseEffectiveRate

Returns the effective rate at which the specified time base is moving relative to its master clock.

```
Fixed GetTimeBaseEffectiveRate (
    TimeBase    tb );
```


tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (II–1119) function.

function result The effective rate at which the time base specified by `tb` is moving relative to its master clock.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Time Base Values” (V–2760)

Related Java Methods

`quicktime.std.clocks.TimeBase.getEffectiveRate()`

GetTimeBaseFlags

Obtains the control flags of a time base.

```
long GetTimeBaseFlags (
    TimeBase  tb );
```

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II–1119).

function result Control flags (see below). Unused flags are set to 0.

Function Result Constants

`loopTimeBase`

Indicates whether or not the time base loops. If this flag is set to 1 and the rate is positive, the time base loops back and restarts from its start time when it reaches its stop time. If this flag is set to 1 and the rate is negative, the time base loops to its stop time. If the flag is set to 0, the movie stops when it reaches the end.

`palindromeLoopTimeBase`

If this flag is set to 1, it indicates that the time base loops in a palindromic fashion. Palindromic looping causes a time base to move alternately forward and backward.

Version Notes

Introduced in QuickTime 3 or earlier.

GetTimeBaseMasterClock

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Time Base Values” (V–2760)

Related Java Methods

`quicktime.std.clocks.TimeBase.getFlags()`

See Also

For the corresponding set function, see `SetTimeBaseFlags` (III–1648).

GetTimeBaseMasterClock

Determines the clock component that is assigned to a time base.

```
ComponentInstance GetTimeBaseMasterClock (
    TimeBase      tb );
```

`tb`

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (II–1119) function.

function result A reference to a component instance. If a clock component is not assigned to the time base, the returned reference is `NIL`. In this case, the time base relies on another time base for its time source. Use `GetTimeBaseMasterTimeBase` (I–517) to obtain the time base reference to that master time base.

Discussion

This function returns a reference to a component instance of the clock component that provides a time source to the specified time base. Every time base derives its time from either a clock component or from another time base. If a time base derives its time from a clock component, use this function to obtain the component instance of the clock component.

Special Considerations

The Component Manager allows a single component to serve multiple client applications at the same time. Each client application has a unique connection to the component, identified by a component instance. Don’t close this connection; the time base is using it to maintain its time source.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Disposing of Time Bases” (V–2759)

Related Java Methods

`quicktime.std.clocks.TimeBase.getMasterClock()`

See Also

For the corresponding set function, see `SetTimeBaseMasterClock` (III–1649).

GetTimeBaseMasterTimeBase

Determines the master time base that is assigned to a time base.

```
TimeBase GetTimeBaseMasterTimeBase (
    TimeBase    tb );
```

`tb`

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II–1119).

function result A time base. If a master time base is not assigned to the time base, this function sets the returned reference to `NIL`. In this case, the time base relies on a clock component for its time source. Use `GetTimeBaseMasterClock` (I–516) to obtain the component instance reference to that clock component.

Discussion

This function returns a reference to the master time base that provides a time source to this time base. A time base derives its time from either a clock component or from another time base. If a time base derives its time from another time base, use this function to obtain the identifier for that master time base.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Disposing of Time Bases” (V–2759)

Related Java Methods

`quicktime.std.clocks.TimeBase.getMasterTimeBase()`

See Also

For the corresponding set function, see `SetTimeBaseMasterTimeBase` (III–1650).

GetTimeBaseRate

GetTimeBaseRate

Retrieves the rate of a time base.

```
Fixed GetTimeBaseRate (  
    TimeBase    tb );
```

tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (II–1119) function.

function result The time base’s rate. This rate value may be nonzero even if the time base has stopped, because it has reached its stop time. Rates may be set to negative values, which cause time to move backward for the time base.

Discussion

This function returns the current rate of the time base as a fixed-point number.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Time Base Values” (V–2760)

Related Java Methods

`quicktime.std.clocks.TimeBase.getRate()`

See Also

For the corresponding set function, see `SetTimeBaseRate` (III–1651).

GetTimeBaseStartTime

Determines the start time of a time base.

```
TimeValue GetTimeBaseStartTime (  
    TimeBase    tb,  
    TimeScale    s,  
    TimeRecord  *tr );
```

tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (II–1119) function.

s

The time scale in which to return the start time.

tr

A pointer to a **time structure** that is to receive the start time. This is an optional parameter. If you don't want the time value represented in a time structure, set this parameter to NIL.

function result The time base's start time.

Discussion

This function returns a time value that contains the start time for the specified time base in the specified time scale. The function returns this value even if you specify a **time structure** with the tr parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Time Base Values” (V–2760)

Related Java Methods

```
quicktime.std.clocks.TimeBase.getStartTime(),
quicktime.std.clocks.TimeBase.getTRStartTime()
```

See Also

For the corresponding set function, see `SetTimeBaseStartTime` (III–1652).

GetTimeBaseStatus

Determines when the current time of a time base would fall outside of the range of values specified by the time base's start and stop times.

```
long GetTimeBaseStatus (
    TimeBase      tb,
    TimeRecord     *unpinnedTime );
```

tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (II–1119) function.

unpinnedTime

A pointer to a **time structure** that is to receive the current time of the time base. Note that this time value may be outside the range of values specified by the start and stop times of the time base.

GetTimeBaseStopTime

function result Status flags (see below).

Function Result Constants

`timeBaseBeforeStartTime`

Indicates that the time value represented by the contents of the **time structure** referred to by the `unpinnedTime` parameter lies before the start time of the time base. The Movie Toolbox sets this flag to 1 if the current time is before the start time of the time base.

`timeBaseAfterStopTime`

Indicates that the time value represented by the contents of the **time structure** referred to by the `unpinnedTime` parameter lies after the stop time of the time base. The Movie Toolbox sets this flag to 1 if the current time is after the stop time of the time base.

Discussion

The status information returned by this function allows you to determine when the current time of a time base would fall outside of the range of values specified by the start and stop times of the time base. This can happen when a time base relies on a master time base or when its time has reached the stop time.

Special Considerations

This function returns no error codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Time Base Values” (V-2760)

Related Java Methods

`quicktime.std.clocks.TimeBase.getStatus()`

See Also

See `GetTimeBaseFlags` (I-515).

GetTimeBaseStopTime

Determines the stop time of a time base.

```
TimeValue GetTimeBaseStopTime (
    TimeBase      tb,
    TimeScale     s,
    TimeRecord    *tr );
```

tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (II–1119) function.

s

The time scale in which to return the stop time.

tr

A pointer to a **time structure** that is to receive the stop time. This is an optional parameter. If you don't want the time value represented in a time structure, set this parameter to `NIL`.

function result The time base's stop time.

Discussion

This function returns a time value that contains the stop time for the specified time base in the specified time scale. The function returns this value even if you specify a **time structure** with the `tr` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Time Base Values” (V–2760)

Related Java Methods

`quicktime.std.clocks.TimeBase.getStopTime()`,
`quicktime.std.clocks.TimeBase.getTRStopTime()`

See Also

For the corresponding set function, see `SetTimeBaseStopTime` (III–1652).

GetTimeBaseTime

Obtains the current time value from a time base.

```
TimeValue GetTimeBaseTime (
    TimeBase      tb,
    TimeScale     s,
    TimeRecord    *tr );
```

tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (II–1119) function.

GetTrackAlternate

s

The time scale in which to return the current time value. Set this parameter to 0 to retrieve the time in the preferred time scale of the time base.

tr

A pointer to a **time structure** that is to receive the current time value. This is an optional parameter. If you don't want the time value represented in a time structure, set this parameter to NIL.

function result The time base's current time.

Discussion

This function returns a time value that contains the current time from the specified time base in the specified time scale. The function returns this value even if you specify a **time structure** with the tr parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Working With Time Base Values” (V–2760)

Related Java Methods

quicktime.std.clocks.TimeBase.getTime(),

quicktime.std.clocks.TimeBase.getTRTime()

See Also

For the corresponding set function, see SetTimeBaseTime (III–1653).

GetTrackAlternate

Determines all the tracks in an alternate group.

```
Track GetTrackAlternate (  
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as NewMovieTrack (II–1092) and GetMovieTrack (I–497).

function result The track identifier of the next track in the group.

Discussion

This function returns the track identifier of the next track in the group. Because the alternate group list is circular, you must specify a different track in the group each

time you call this function. You have retrieved all the tracks in the group when the function returns the track identifier that you supplied the first time you called `GetTrackAlternate`. If there is only one track in an alternate group, or if the track you specify does not belong to a group, this function returns the track identifier you supply.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Alternate Tracks” (V–2738)

Related Java Methods

`quicktime.std.movies.Track.getAlternate()`

See Also

For the corresponding set function, see `SetTrackAlternate` (III–1656).

GetTrackBoundsRgn

Lets the media limit the size of a track boundary rectangle.

```
RgnHandle GetTrackBoundsRgn (
    Track    theTrack );
```

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result A handle to the region limited by the media.

Discussion

Because the media limits the size of the track boundary rectangle, the region returned by `GetTrackBoundsRgn` may not be rectangular and may be smaller than the track boundary region.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.qd.Region.fromTrackBounds()`,
`quicktime.std.movies.Track.getBoundsRgn()`

GetTrackClipRgn

See Also

Movie Toolbox uses the `MediaGetSrcRgn` (II–865) function to determine whether your media has an irregularly shaped display area. If your media is displayed in a nonrectangular area, or if your media uses only a portion of the track rectangle, you can use this function to report that fact to the toolbox.

GetTrackClipRgn

Determines the clipping region of a track.

```
RgnHandle GetTrackClipRgn (  
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result A handle to the track’s clipping region.

Discussion

This function allocates the region and returns a handle to the region. Your application must dispose of this region when you are done with it. If the function could not satisfy your request or if there is no clipping region defined for the track, it sets the returned handle to `NIL`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.qd.Region.fromTrackClip()`, `quicktime.std.movies.Track.getClipRgn()`

See Also

For the corresponding set function, see `SetTrackClipRgn` (III–1656).

GetTrackCreationTime

Returns a track’s creation date and time.

```
long GetTrackCreationTime (  
    Track    theTrack );
```

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result The track’s creation date and time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Determining Movie Creation and Modification Time” (V–2741)

Related Java Methods

`quicktime.std.movies.Track.getCreationTime()`

GetTrackDataSize

Determines the size, in bytes, of the sample data in a segment of a track.

```
long GetTrackDataSize (
    Track      theTrack,
    TimeValue  startTime,
    TimeValue  duration );
```

`theTrack`

The track for this operation. You obtain this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

`startTime`

A time value specifying the starting point of the segment.

`duration`

A time value that specifies the duration of the segment.

function result The size, in bytes, of the sample data in a segment of a track.

Discussion

This function counts each use of a sample. That is, if a track uses a given sample more than once, the size of that sample is included in the returned size value one time for each use. Consequently, the returned size is greater than or equal to the actual size of the track’s sample data.

Version Notes

Introduced in QuickTime 3 or earlier.

GetTrackDataSize64

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Samples” (V–2742)

Related Java Methods

`quicktime.std.movies.Track.getDataSize()`

See Also

See `GetTrackDataSize64` (I–526).

GetTrackDataSize64

Provides a 64-bit version of `GetTrackDataSize` (I–525).

```
OSErr GetTrackDataSize64 (
    Track          theTrack,
    TimeValue      startTime,
    TimeValue      duration,
    wide           *dataSize );
```

theTrack

A track identifier. Your application obtains this identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

startTime

A time value specifying the starting point of the segment.

duration

A time value that specifies the duration of the segment.

dataSize

The size, in bytes, of the sample data in a segment of a track. This function counts each use of a sample. That is, if a track uses a given sample more than once, the size of that sample is included in the returned size value one time for each use. Consequently, the returned size is greater than or equal to the actual size of the track’s sample data.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The only difference between this function and `GetTrackDataSize` (I–525) is that size of the sample data is returned as a 64-bit integer in the *dataSize* parameter instead of as a 32-bit integer returned by the function.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

See Also

See `GetTrackDataSize` (I-525).

GetTrackDimensions

Determines a track's source rectangle.

```
void GetTrackDimensions (
    Track    theTrack,
    Fixed    *width,
    Fixed    *height );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` and `GetMovieTrack` (I-497).

width

A pointer to a fixed-point number. The Movie Toolbox returns the width, in pixels, of the track's rectangle. This value corresponds to the x coordinate of the lower-right corner of the track's rectangle.

height

A pointer to a fixed-point number. The Movie Toolbox returns the height, in pixels, of the track's rectangle. This value corresponds to the y coordinate of the lower-right corner of the track's rectangle.

function result You can access this function's error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Working With Movie Spatial Characteristics" (V-2728)

Related Java Methods

`quicktime.std.movies.Track.getDimensions()`

GetTrackDisplayBoundsRgn

See Also

For the corresponding set function, see `SetTrackDimensions` (III–1657).

GetTrackDisplayBoundsRgn

Determines the region a track occupies in a movie’s graphics world.

```
RgnHandle GetTrackDisplayBoundsRgn (  
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result A handle to the region the specified track occupies in a movie’s graphics world.

Discussion

This function allocates the region and returns a handle to the region. If the track does not have a spatial representation at the current movie time, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to `NIL`.

Special Considerations

Your application must dispose of the returned region when you are done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

```
quicktime.qd.Region.fromTrackDisplay(),  
quicktime.std.movies.Track.getDisplayBoundsRgn()
```

GetTrackDisplayMatrix

Returns a matrix that is the concatenation of all matrices currently affecting the track’s location, scaling, and so on, including the movie’s matrix, the track’s matrix, and the modifier matrix.

```
OSErr GetTrackDisplayMatrix (
```

```
Track          theTrack,
MatrixRecord   *matrix );
```

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

`matrix`

A pointer to a matrix structure.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Since modifier information is passed between tracks at `MoviesTask` (II–973) time, the information returned by this call represents the matrix in effect at the last `MoviesTask` call.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V–2722)

Related Java Methods

`quicktime.std.movies.Track.getDisplayMatrix()`

See Also

To determine the entire clip of a track at the current time, use `GetTrackDisplayBoundsRgn` (I–528). The results of `GetTrackDisplayBoundsRgn` take into account any clip regions provided by modifier tracks.

GetTrackDuration

Returns the duration of a track.

```
TimeValue GetTrackDuration (
    Track      theTrack );
```

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result The duration of the specified track, expressed in the time scale of the

GetTrackEditRate

movie that contains the track.

Discussion

The duration corresponds to the ending time of the track in the movie's time coordinate system (remember that all tracks start at movie time 0).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Working With Track Time" (V-2733)

Related Java Methods

`quicktime.std.movies.Track.getDuration()`

GetTrackEditRate

Returns the rate of the track edit of a specified track at an indicated time.

```
Fixed GetTrackEditRate (
    Track          theTrack,
    TimeValue      atTime );
```

theTrack

The track identifier for which the rate of a track edit (at the time given in the *atTime* parameter) is to be determined.

atTime

Indicates a time value at which the rate of a track edit (of a track identified in the parameter *theTrack*) is to be determined.

function result The rate of the track edit of the specified track at the specified time.

Discussion

This function is useful if you are stepping through track edits directly in your application or if you are a client of QuickTime's base media handler.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Editing Tracks" (V-2750)

Related Java Methods

`quicktime.std.movies.Track.getEditRate()`

GetTrackEnabled

Determines whether a track is currently enabled.

```
Boolean GetTrackEnabled (
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` and `GetMovieTrack` (I–497).

function result TRUE if the specified track is currently enabled, FALSE otherwise.

Discussion

The Movie Toolbox services only enabled tracks.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Disabling Movies and Tracks” (V–2724)

Related Java Methods

`quicktime.std.movies.Track.setEnabled()`

See Also

For the corresponding set function, see `SetTrackEnabled` (III–1658).

GetTrackID

Determines a track’s unique track ID value.

```
long GetTrackID (
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result The specified track’s unique track ID value.

Version Notes

Introduced in QuickTime 3 or earlier.

GetTrackLayer

Programming Info

C interface file: `Movies.h`

Programming summary: “Locating a Movie’s Tracks and Media Structures” (V–2736)

Related Java Methods

`quicktime.std.movies.Track.getID()`

GetTrackLayer

Retrieves a track’s layer.

```
short GetTrackLayer (
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result The specified track’s layer number.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Track.getLayer()`

See Also

For the corresponding set function, see `SetTrackLayer` (III–1660).

GetTrackLoadSettings

Retrieves a track’s preload information.

```
void GetTrackLoadSettings (
    Track        theTrack,
    TimeValue    *preloadTime,
    TimeValue    *preloadDuration,
    long         *preloadFlags,
    long         *defaultHints );
```

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

`preloadTime`

Specifies a field to receive the starting point of the portion of the track to be preloaded. The toolbox returns a value of –1 if the entire track is to be preloaded.

`preloadDuration`

Specifies a field to receive the amount of the track to be preloaded, starting from the time specified in the `preloadTime` parameter. If the entire track is to be preloaded, this value is ignored.

`preloadFlags`

Specifies a field to receive the flags (see below) that control when the toolbox preloads the track.

`defaultHints`

Specifies a field to receive the playback hints for the track.

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

preloadFlags Constants

`preloadAlways`

Specifies that the toolbox always preloads this track.

`preloadOnlyIfEnabled`

Specifies that the toolbox preloads this track only when the track is enabled.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V–2722)

Related Java Methods

`quicktime.std.movies.Track.getLoadSettings()`

See Also

For the corresponding set function, see `SetTrackLoadSettings` (III–1661).

GetTrackMatrix

Retrieves a track’s transformation matrix.

GetTrackMatte

```
void GetTrackMatrix (
    Track      theTrack,
    MatrixRecord *matrix );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` and `GetMovieTrack` (I–497).

matrix

A pointer to a `MatrixRecord` (IV–2304) structure. The `GetTrackMatrix` function returns the track’s matrix into the structure referred to by this parameter.

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Track.getMatrix()`

See Also

For the corresponding set function, see `SetTrackMatrix` (III–1662). The Movie Toolbox provides a number of functions that allow you to manipulate track matrices. See `SetIdentityMatrix` (III–1593) for information about these functions.

GetTrackMatte

Retrieves a copy of a track’s matte.

```
PixMapHandle GetTrackMatte (
    Track      theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result A handle to a `PixMap` (IV–2332) structure that represents the specified track’s matte. If the function could not satisfy your request, it sets the returned handle to `NIL`.

Discussion

The matte defines which of the track's pixels are displayed in a movie, and it is valid for the entire duration of the movie.

Special Considerations

You should use `DisposeMatte` (I–267) to dispose of the matte when you are finished with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

See Also

For the corresponding set function, see `SetTrackMatte` (III–1663).

GetTrackMedia

Determines the media that contains a track's sample data.

```
Media GetTrackMedia (
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result The media identifier for the media that contains the track's sample data. If the function could not locate the media, it sets this returned value to `NIL`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Locating a Movie's Tracks and Media Structures” (V–2736)

Related Java Methods

```
quicktime.std.movies.media.Media.fromTrack(),
quicktime.std.movies.media.Media.getTrackMedia()
```

GetTrackModificationTime

GetTrackModificationTime

Returns a track's modification date and time.

```
long GetTrackModificationTime (  
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result The specified track's modification date and time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Determining Movie Creation and Modification Time” (V–2741)

Related Java Methods

`quicktime.std.movies.Track.getModificationTime()`

GetTrackMovie

Determines the movie that contains a specified track.

```
Movie GetTrackMovie (  
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` and `GetMovieTrack` (I–497).

function result The identifier of the movie that contains the track.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Locating a Movie's Tracks and Media Structures” (V–2736)

Related Java Methods

```
quicktime.std.movies.Track.getMovie()
```

GetTrackMovieBoundsRgn

Determines the region the track occupies in a movie's boundary region.

```
RgnHandle GetTrackMovieBoundsRgn (  
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result A handle to the region the specified track occupies in its movie's boundary region. If the track does not have a spatial representation at the current movie time, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to `NIL`.

Discussion

This function determines the region by applying the track's clipping region and matrix. This region is valid only for the current movie time. The function allocates the region and returns a handle to it.

Special Considerations

Your application must dispose of the returned region when you are done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

```
quicktime.qd.Region.fromTrackMovieBounds(),  
quicktime.std.movies.Track.getMovieBoundsRgn()
```

GetTrackNextInterestingTime

Searches for times of interest in a track.

```
void GetTrackNextInterestingTime (
```

GetTrackNextInterestingTime

```
Track          theTrack,  
short          interestingTimeFlags,  
TimeValue      time,  
Fixed          rate,  
TimeValue      *interestingTime,  
TimeValue      *interestingDuration );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

interestingTimeFlags

Contains flags (see below) that determine the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit`, `nextTimeTrackEdit` and `nextTimeSyncSample` flags to 1. Set unused flags to 0.

time

Specifies a time value that establishes the starting point for the search. This time value must be expressed in the movie's time scale.

rate

The search direction. Negative values cause the Movie Toolbox to search backward from the starting point specified in the `time` parameter. Other values cause a forward search.

interestingTime

A pointer to a time value. The Movie Toolbox returns the first time value it finds that meets the search criteria specified in the `flags` parameter. This time value is in the movie's time scale. If there are no times that meet the search criteria you specify, the Movie Toolbox sets this value to –1. Set this parameter to `NIL` if you are not interested in this information.

interestingDuration

A pointer to a time value. The Movie Toolbox returns the duration of the interesting time. This time value is in the movie's time coordinate system. Set this parameter to `NIL` if you don't want this information; in this case, the function works more quickly.

interestingTimeFlags Constants

nextTimeMediaSample

Searches for the next sample in the track's media. Set this flag to 1 to search for the next sample.

nextTimeMediaEdit

Searches for the next group of samples in the track's media. Set this flag to 1 to search for the next group of samples.

nextTimeTrackEdit

Searches for the media sample that corresponds to the next entry in a track's media edit list. The end of the track is considered an empty edit. Set this flag to 1 to search for the next track edit.

nextTimeSyncSample

Searches for the next **sync sample** in the track's media. Set this flag to 1 to search for the next sync sample.

nextTimeEdgeOK

Instructs the Movie Toolbox that you are willing to receive information about elements that begin or end at the time specified by the `time` parameter. Set this flag to 1 to accept this information. When this flag is set, the function returns valid information about the beginning and end of the track.

nextTimeIgnoreActiveSegment

Instructs the Movie Toolbox to look outside of the active segment for samples that meet the search criteria. Set this flag to 1 to search outside of the active segment.

Discussion

Some compression algorithms conserve space by eliminating duplication between consecutive frames in a sample. In this case, **sync samples** don't rely on preceding frames for content. You can access error returns from this function through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494). See "Error Codes" (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Finding Interesting Times" (V-2735)

Related Java Methods

`quicktime.std.movies.Track.getNextInterestingTime()`

GetTrackOffset

Determines the time difference between the start of a track and the start of the movie that contains the track.

```
TimeValue GetTrackOffset (
    Track    theTrack );
```

GetTrackPict

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result The time difference between the start of the specified track and the start of the movie that contains the track.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Track Time” (V–2733)

Related Java Methods

`quicktime.std.movies.Track.getOffset()`

See Also

For the corresponding set function, see `SetTrackOffset` (III–1664).

GetTrackPict

Creates a QuickDraw picture from a specified track at a specified time.

```
PicHandle GetTrackPict (
    Track        theTrack,
    TimeValue    time );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

time

The time at which the image is taken.

function result A handle to the specified picture. If the function could not create the picture, the returned handle is set to `NIL`.

Special Considerations

Your application must dispose of the returned picture handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Generating Pictures From Movies” (V–2725)

Related Java Methods

`quicktime.qd.Pict.fromTrack()`, `quicktime.std.movies.Track.getPict()`

See Also

`GetTrackPict` is similar to `GetMoviePict` (I-483), except that `GetTrackPict` uses only the specified track to create the picture.

GetTrackReference

Retrieves the track identifier contained in an existing track reference.

```
Track GetTrackReference (
    Track    theTrack,
    OSType   refType,
    long     index );
```

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II-1092) and `GetMovieTrack` (I-497).

refType

The type of reference; see “Data References” (IV-2661).

index

The index value of the reference found. You obtain this index value when you create the track reference.

function result The track identifier for the specified track. If the toolbox cannot locate the track reference corresponding to your specifications, it returns a value of NIL.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Track References” (V-2737)

Related Java Methods

`quicktime.std.movies.Track.getReference()`

See Also

For the corresponding set function, see `SetTrackReference` (III-1665).

GetTrackReferenceCount

GetTrackReferenceCount

Determines how many track references of a given type exist for a track.

```
long GetTrackReferenceCount (
    Track    theTrack,
    OSType   refType );
```

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

refType

The type of reference; see “Data References” (IV–2661). The toolbox determines the number of track references of this type.

function result A long integer that specifies the number of track references of the specified type in the track. If there are no references of the type you have specified, the function returns a value of 0.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Track References” (V–2737)

Related Java Methods

`quicktime.std.movies.Track.getReferenceCount()`

GetTrackSegmentDisplayBoundsRgn

Determines the region a track occupies in a movie’s graphics world during a specified segment.

```
RgnHandle GetTrackSegmentDisplayBoundsRgn (
    Track    theTrack,
    TimeValue time,
    TimeValue duration );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

time

The starting time of the track segment to consider. This time value must be expressed in the movie's time coordinate system. The duration parameter specifies the length of the segment.

duration

The length of the segment to consider. Set this parameter to 0 to consider an instant in time.

function result

A handle to the region the specified track occupies in its movie's graphics world during a specified segment. If the track does not have a spatial representation during the specified segment, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to NIL.

Discussion

This function allocates the region and returns a handle to it. This region is valid for the specified segment.

Special Considerations

Your application must dispose of the returned region when you are done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Working With Movie Spatial Characteristics" (V-2728)

Related Java Methods

```
quicktime.qd.Region.fromTrackSegment(),  
quicktime.std.movies.Track.getSegmentDisplayBoundsRgn()
```

GetTrackSoundLocalizationSettings

Returns a handle to a copy of the current 3D sound settings for a specified track.

```
OSErr GetTrackSoundLocalizationSettings (  
    Track      theTrack,  
    Handle     *settings );
```

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II-1092) and `GetMovieTrack` (I-497).

GetTrackStatus

settings

A handle to a copy of the current 3D sound settings for a specified track, in the format of an `SSpLocalizationData` (IV–2460) record. If there are no 3D sound settings, the returned handle is set to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Special Considerations

The caller of this function is responsible for disposing of the returned handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Track Sound” (V–2740)

Related Java Methods

`quicktime.std.movies.Track.getSoundLocalizationSettings()`,

`quicktime.util.QTHandle.fromTrack()`

See Also

For the corresponding set function, see `SetTrackSoundLocalizationSettings` (III–1666).

GetTrackStatus

Returns the value of the last error the media encountered while playing a specified track.

```
ComponentResult GetTrackStatus (
    Track      theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from `GetMovieStatus` (I–494).

function result `GetTrackStatus` returns the last error encountered for the specified track; see “Error Codes” (IV–2663). If the component does not find any errors, the result is set to `noErr`.

Discussion

This function returns information about errors that are encountered during the processing associated with `MoviesTask` (II–973). These errors typically reflect playback problems, such as low-memory conditions. This function returns the last error encountered for the specified track. The media clears this error code when it detects that the error has been corrected.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movies and Your Event Loop” (V–2720)

Related Java Methods

`quicktime.std.movies.Track.getStatus()`

See Also

See `MoviesTask` (II–973).

GetTrackUsage

Determines whether a track is used in a movie, its **preview**, its **poster**, or a combination of these.

```
long GetTrackUsage (
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result Track usage flags (see below). These flags may be combined.

Function Result Constants

`trackUsageInMovie`

If this flag is set to 1, the track is used in its movie.

`trackUsageInPreview`

If this flag is set to 1, the track is used in the **preview**.

`trackUsageInPoster`

If this flag is set to 1, the track is used in the **poster**.

Version Notes

Introduced in QuickTime 3 or earlier.

GetTrackUserData

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V–2718)

Related Java Methods

`quicktime.std.movies.Track.getUsage()`

See Also

Your application can specify how a track is used by calling `SetTrackUsage` (III–1668).

GetTrackUserData

Obtains access to a track’s **user data list**.

```
UserData GetTrackUserData (  
    Track    theTrack );
```

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result A reference to the specified track’s user data. If the function could not locate the track’s user data, it sets this returned value to `NIL`.

Discussion

This function returns a reference to the track’s **user data list**, which is valid until you dispose of the track. When you save the track, the Movie Toolbox saves the user data as well.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

```
quicktime.std.movies.media.UserData.fromTrack(),  
quicktime.std.movies.Track.getUserData()
```

See Also

You can use `GetUserData` (I–547), `AddUserData` (I–44), and `RemoveUserData` (III–1459) to manipulate the contents of the **user data list**. If the data list contains text data, you can use `GetUserDataText` (I–549), `AddUserDataText` (I–45), and `RemoveUserDataText` (III–1460) to work with its contents.

GetTrackVolume

Returns a track's current volume setting.

```
short GetTrackVolume (
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result The specified track's current volume setting. The values returned in the high and low words range from 0x0000 (silence) to 0x0100 (full volume). You can use constants (see below) to test for full volume and no volume.

Function Result Constants

kFullVolume

Full volume (constant value is 1.0).

kNoVolume

No volume (constant value is 0.0).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Sound Volume” (V–2732)

Related Java Methods

`quicktime.std.movies.Track.getVolume()`

See Also

For the corresponding set function, see `SetTrackVolume` (III–1669).

GetUserData

Returns a specified user data item.

```
OSErr GetUserData (
    UserData    theUserData,
    Handle      data,
    OSType      udType,
    long        index );
```

GetUserDataItem

theUserData

The **user data list** for this operation. You obtain this list reference by calling the `GetMovieUserData` (I–499), `GetTrackUserData` (I–546), or `GetMediaUserData` (I–456) function.

data

A handle that is to receive the data from the specified item. `GetUserData` resizes this handle as appropriate to accommodate the item. Your application is responsible for releasing this handle when you are done with it. Set this parameter to `NIL` if you don’t want to retrieve the user data item. This can be useful if you want to verify that a user data item exists, but you don’t need to work with the item’s contents.

udType

The item’s type value; see “User Data Identifiers” (IV–2696).

index

The item’s index value. This parameter must specify an item in the **user data list** identified by the parameter *theUserData*.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

`quicktime.std.movies.media.UserData.getData()`

GetUserDataItem

Returns a specified user data item.

```
OSErr GetUserDataItem (
    UserData    theUserData,
    void        *data,
    long        size,
    OSType      udType,
    long        index );
```

theUserData

The **user data list** for this operation. You obtain this list reference by calling the `GetMovieUserData` (I–499), `GetTrackUserData` (I–546), or `GetMediaUserData` (I–456).

data

A pointer that is to receive the data from the specified item.

size

The size of the item.

udType

The item’s type value; see “User Data Identifiers” (IV–2696).

index

The item’s index value. This parameter must specify an item in the **user data list** identified by the parameter *theUserData*.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

`quicktime.std.movies.media.UserData.getDataItem()`

See Also

For the corresponding set function, see `SetUserDataItem` (III–1673).

GetUserDataText

Retrieves language-tagged text from an item in a **user data list**.

```
OSErr GetUserDataText (
    UserData    theUserData,
    Handle      data,
    OSType      udType,
    long        index,
    short       itlRegionTag );
```

GoToBeginningOfMovie

`theUserData`

The **user data list** for this operation. You obtain this list reference by calling the `GetMovieUserData` (I–499), `GetTrackUserData` (I–546), or `GetMediaUserData` (I–456) function.

`data`

A handle that is to receive the data. The `GetUserDataText` function resizes this handle as appropriate. Your application must dispose of the handle when you are done with it.

`udType`

The item’s type value; see “User Data Identifiers” (IV–2696).

`index`

The item’s index value. This parameter must specify an item in the **user data list** identified by the parameter `theUserData`.

`itlRegionTag`

The language code of the text to be retrieved. See “Localization Codes” (IV–2673).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You specify the **user data list** and item, and the item’s type value and language code. The Movie Toolbox retrieves the specified text from the user data item.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

`quicktime.std.movies.media.UserData.getText()`

GoToBeginningOfMovie

Repositions a movie to play from its start.

```
void GoToBeginningOfMovie (
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

Discussion

If the movie is in **preview** mode, the function goes to the start of the preview segment of the movie. In all other cases, this function goes to the start of the movie, where the movie time value is 0. You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Controlling Movie Playback” (V–2717)

Related Java Methods

`quicktime.std.movies.Movie.goToBeginning()`

GoToEndOfMovie

Repositions a movie to play from its end.

```
void GoToEndOfMovie (
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Controlling Movie Playback” (V–2717)

Related Java Methods

`quicktime.std.movies.Movie.goToEnd()`

GraphicsExportCanTranscode

Asks whether the current graphics export operation should be performed by transcoding.

```
ComponentResult GraphicsExportCanTranscode (  
    GraphicsExportComponent    ci,  
    Boolean                    *canTranscode );
```

ci

The component instance that identifies your connection to the graphics exporter component.

canTranscode

Points to a `Boolean` to receive the answer. `TRUE` means that the current export operation should be performed by transcoding, `FALSE` that it should not.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Graphics exporters may be able to transcode from some inputs and not from others. For instance, the JPEG graphics exporter is able to transcode compressed JPEG streams, but not other kinds of compressed data. The **base graphics exporter** makes this call to the format-specific graphics exporter to ask whether the current export operation should be done by transcoding. If the format-specific exporter replies that it should, the base exporter calls `GraphicsExportDoTranscode` (I–555) to do so. If the answer is no, then the format-specific exporter will not be able to transcode.

Special Considerations

This function is used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call it. Format-specific exporters may delegate this call, in which case the **base graphics exporter**’s implementation gives a reply of `FALSE`.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Internal Graphics Export Routines” (V–2883)

See Also

Format-specific exporters implementing `GraphicsExportCanTranscode` should call `GraphicsExportMayExporterReadInputData` (I–585).

GraphicsExportCanUseCompressor

Asks whether to use a compressor in a graphics export operation.

```
ComponentResult GraphicsExportCanUseCompressor (
    GraphicsExportComponent    ci,
    Boolean                    *canUseCompressor,
    void                       *codecSettingsAtomContainerPtr );
```

ci

The component instance that identifies your connection to the graphics exporter component.

canUseCompressor

A Boolean variable to receive the answer.

codecSettingsAtomContainerPtr

A pointer to a QTAtomContainer variable. If the *canUseCompressor* parameter returns TRUE, the format-specific exporter should create a new QuickTime atom container with information about the compression operation and return it here.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Discussion

The **base graphics exporter** makes this call of the format-specific graphics exporter to ask whether the current export operation should be done by using an image compressor. If the answer is TRUE, the format-specific exporter must also create and return an atom container. This atom container must contain a **big-endian** 'vide' (IV–2610) atom with at least a child atom of type 'spt1' (IV–2586) containing a SCSpatialSettings (IV–2423) record specifying which compressor to use, the depth, and the spatial quality.

Special Considerations

This function is used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call it. Format-specific exporters may delegate this call, in which case the **base graphics exporter**'s implementation gives a reply of FALSE.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Internal Graphics Export Routines” (V–2883)

GraphicsExportDoExport

GraphicsExportDoExport

Performs a graphics export operation.

```
ComponentResult GraphicsExportDoExport (
    GraphicsExportComponent    ci,
    unsigned long               *actualSizeWritten );
```

ci

The component instance that identifies your connection to the graphics exporter component.

actualSizeWritten

Points to a variable to receive the number of bytes written. If you are not interested in this information, pass NIL.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Before calling this function , you must specify an input image, using one of the `GraphicsExportSetInput...` functions, and a destination for the output image file, using one of the `GraphicsExportSetOutput...` functions.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Setting the Input and Output of a Graphics Exporter Instance” (V–2883)

GraphicsExportDoStandaloneExport

Performs a standalone graphics export operation.

```
ComponentResult GraphicsExportDoStandaloneExport (
    GraphicsExportComponent    ci );
```

ci

The component instance that identifies your connection to the graphics exporter component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If both `GraphicsExportCanTranscode` (I–552) and `GraphicsExportCanUseCompressor` (I–553) reply `FALSE`, the **base graphics exporter** makes this call of the format-specific exporter to perform the export.

Special Considerations

This function is used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Internal Graphics Export Routines” (V–2883)

See Also

The format-specific exporter may call `GraphicsExportGetInputImageDescription` (I–570), `GraphicsExportGetInputImageDimensions` (I–571), and `GraphicsExportGetInputImageDepth` (I–569) to find out about the input image. It should allocate a `PixMap` (IV–2332) structure and call `GraphicsExportDrawInputImage` (I–557) to draw portions of the input image into it. The format-specific exporter should call `GraphicsExportWriteOutputData` (I–611) to write the image data.

GraphicsExportDoTranscode

Performs a graphics export operation by transcoding.

```
ComponentResult GraphicsExportDoTranscode (
    GraphicsExportComponent  ci );
```

ci

The component instance that identifies your connection to the graphics exporter component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The **base graphics exporter** makes this call of the format-specific graphics exporter to perform a transcoding export. This function should call `GraphicsExportGetInputDataSize` (I–564) and `GraphicsExportReadInputData` (I–586) to measure and read the input image data, and `GraphicsExportWriteOutputData` (I–611) to write the output image file.

GraphicsExportDoUseCompressor

Special Considerations

This function is used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Internal Graphics Export Routines” (V–2883)

GraphicsExportDoUseCompressor

Performs a graphics export operation with compression.

```
ComponentResult GraphicsExportDoUseCompressor (
    GraphicsExportComponent    ci,
    void                      *codecSettingsAtomContainer,
    ImageDescriptionHandle     *outDesc );
```

ci

The component instance that identifies your connection to the graphics exporter component.

codecSettingsAtomContainer

An atom container returned by `GraphicsExportCanUseCompressor` (I–553).

outDesc

Points to an image description handle to receive an `ImageDescription` (IV–2285) structure describing the compressed image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The **base graphics exporter** makes this call to perform a compressing export.

Special Considerations

This function is used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call it. Format-specific exporters will normally delegate this call, unless they implement export to a container format like PICT or QuickTime Image. In that case, they will wrap the base exporter’s implementation in one that forms the container about the compressed data.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Internal Graphics Export Routines” (V-2883)

GraphicsExportDrawInputImage

Draws a rectangular portion of the input image in a graphics export operation.

```
ComponentResult GraphicsExportDrawInputImage (
    GraphicsExportComponent    ci,
    CGrafPtr                   gw,
    GDHandle                    gd,
    const Rect                  *srcRect,
    const Rect                  *dstRect );
```

ci

The component instance that identifies your connection to the graphics exporter component.

gw

A pointer to an offscreen graphics world, color graphics port, or basic graphics port.

gd

A handle to a GDevice (IV-2264) record. If you pass a pointer to an offscreen graphics world in the *gw* parameter, set this parameter to `NIL` because `GraphicsExportDrawInputImage` ignores this parameter and sets the current device to the device attached to the offscreen graphics world.

srcRect

Specifies a portion of the input image.

dstRect

Specifies where in the drawing environment to draw that portion of the input image.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

The *gw* and *gd* parameters specify a drawing environment such as you might pass to `GraphicsExportSetInputGWorld` (I-596). The *srcRect* and *dstRect* boundaries need not be the same width and height; you can use this function to scale the *srcRect* image portion. This would be useful, for example, if you were writing a graphics exporter for a multiple-resolution format.

GraphicsExportGetColorSyncProfile

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing a Graphics Exporter’s Input Image” (V–2889)

See Also

When doing a standalone export, an exporter will typically call either `GraphicsExportGetInputImageDescription` (I–570), or both `GraphicsExportGetInputImageDimensions` (I–571) and `GraphicsExportGetInputImageDepth` (I–569), to determine the image’s bounds and depth, and then allocate an offscreen graphics world and call `GraphicsExportDrawInputImage` to draw portions of the image into that world.

GraphicsExportGetColorSyncProfile

Gets the current value of the ColorSync profile for a graphics export operation.

```
ComponentResult GraphicsExportGetColorSyncProfile (
    GraphicsExportComponent    ci,
    Handle                     *colorSyncProfile );
```

ci

The component instance that identifies your connection to the graphics exporter component.

colorSyncProfile

Points to a variable to receive the ColorSync profile as a newly allocated handle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

The caller is responsible for disposing of the returned handle.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding set function, see `GraphicsExportSetColorSyncProfile` (I–589).

GraphicsExportGetCompressionMethod

Returns the compression method for a graphics export operation.

```
ComponentResult GraphicsExportGetCompressionMethod (
    GraphicsExportComponent    ci,
    long                       *compressionMethod );
```

ci

The component instance that identifies your connection to the graphics exporter component.

compressionMethod

Points to a value to receive the compression method.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding set function, see `GraphicsExportSetCompressionMethod` (I–589).

GraphicsExportGetCompressionQuality

Returns the compression quality value for a graphics export operation.

```
ComponentResult GraphicsExportGetCompressionQuality (
    GraphicsExportComponent    ci,
    CodecQ                     *spatialQuality );
```

ci

The component instance that identifies your connection to the graphics exporter component.

spatialQuality

Points to a variable to receive a quality constant (see below).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

GraphicsExportGetDefaultFileNameExtension

spatialQuality Constants

`codecMinQuality`

The minimum valid value for a CodecQ field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a CodecQ field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding set function, see `GraphicsExportSetCompressionQuality` (I–590).

GraphicsExportGetDefaultFileNameExtension

Returns the suggested file name extension for a graphics export operation.

```
ComponentResult GraphicsExportGetDefaultFileNameExtension (
    GraphicsExportComponent    ci,
    OSType                    *fileNameExtension );
```

`ci`

The component instance that identifies your connection to the graphics exporter component.

`fileNameExtension`

Points to a location to receive the file name extension.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

File name extensions are returned as upper-case **big-endian** four-character codes. For example, the extension `.png` would be returned as `'PNG'` (0x504E4720).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Finding Out About Graphics Export Image Formats” (V–2884)

GraphicsExportGetDefaultFileTypeAndCreator

Returns the suggested file type and creator for a graphics export operation.

```
ComponentResult GraphicsExportGetDefaultFileTypeAndCreator (
    GraphicsExportComponent ci,
    OSType *fileType,
    OSType *fileCreator );
```

ci

The component instance that identifies your connection to the graphics exporter component.

fileType

Points to a location to receive the suggested file type for the image file format. If you don't need this information, pass `NIL`.

fileCreator

Points to a location to receive the suggested file creator for the new image file format. If you don't need this information, pass `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function, along with `GraphicsExportGetDefaultFileNameExtension` (I–560) and `GraphicsExportGetMIMETypeList` (I–577), returns information about the image format supported by a graphics exporter. Format-specific exporters must implement all three of these calls.

Version Notes

Introduced in QuickTime 4.

GraphicsExportGetDepth

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Finding Out About Graphics Export Image Formats” (V–2884)

See Also

See `GraphicsExportGetDefaultFileNameExtension` (I–560) and `GraphicsExportGetMIMETypeList` (I–577).

GraphicsExportGetDepth

Returns the current depth setting for a graphics export operation.

```
ComponentResult GraphicsExportGetDepth (  
    GraphicsExportComponent    ci,  
    long                       *depth );
```

ci

The component instance that identifies your connection to the graphics exporter component.

depth

Points to a variable to receive the depth.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding set function, see `GraphicsExportSetDepth` (I–591).

GraphicsExportGetDontRecompress

Determines whether the original compressed data for a graphics export operation will not be decompressed and recompressed, but be copied through to the output file.

```
ComponentResult GraphicsExportGetDontRecompress (  
    GraphicsExportComponent    ci,
```



```
Boolean                                *dontRecompress );
```

ci

The component instance that identifies your connection to the graphics exporter component.

dontRecompress

Points to a Boolean to receive the recompression setting.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Even though it is not decompressed and recompressed, graphics data may be modified when it is copied through.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding set function, see `GraphicsExportSetDontRecompress` (I–592). To determine whether a particular export setting is available, use `CallComponentCanDo` (I–58).

GraphicsExportGetInputDataReference

Returns the current input data reference for a graphics export operation.

```
ComponentResult GraphicsExportGetInputDataReference (
    GraphicsExportComponent  ci,
    Handle                   *dataRef,
    OSType                   *dataRefType );
```

ci

The component instance that identifies your connection to the graphics exporter component.

dataRef

Points to a variable to receive the data reference handle.

dataRefType

Points to a variable to receive the data reference type.

GraphicsExportGetInputDataSize

function result See “Error Codes” (IV–2663). If the current source is not a data reference, the function returns paramErr. The function returns noErr if there is no error.

Discussion

You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a graphics world, a PixMap (IV–2332) structure, or a piece of compressed data described by an ImageDescription (IV–2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Special Considerations

The caller is responsible for disposing of the returned data reference handle.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding set function, see GraphicsExportSetInputDataReference (I–593). The following other set functions also specify the sources for input images: GraphicsExportSetInputFile (I–594), GraphicsExportSetInputHandle (I–597), GraphicsExportSetInputPtr (I–602), GraphicsExportSetInputGraphicsImporter (I–595), GraphicsExportSetInputPicture (I–599), GraphicsExportSetInputGWorld (I–596), and GraphicsExportSetInputPixmap (I–600).

GraphicsExportGetInputDataSize

Returns the number of bytes of original image data that can be read in a graphics export operation.

```
ComponentResult GraphicsExportGetInputDataSize (
    GraphicsExportComponent    ci,
    unsigned long               *size );
```

ci

The component instance that identifies your connection to the graphics exporter component.

size

Points to a variable to receive the size in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is used by format-specific graphics exporters when transcoding. Applications will not normally need to call this function.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Reading Graphics Exporter Input Data” (V–2889)

GraphicsExportGetInputFile

Returns the current input file for a graphics export operation.

```
ComponentResult GraphicsExportGetInputFile (
    GraphicsExportComponent  ci,
    FSSpec                  *theFile );
```

ci

The component instance that identifies your connection to the graphics exporter component.

theFile

A pointer to the file specification of the file containing the graphics data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. If the current source is not a file, the function returns `paramErr`.

Discussion

You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a graphics world, a `PixMap` (IV–2332) structure, or a piece of compressed data described by an `ImageDescription` (IV–2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or

GraphicsExportGetInputGraphicsImporter

given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding set function, see `GraphicsExportSetInputFile` (I–594). The following other set functions also specify the sources for input images: `GraphicsExportSetInputDataReference` (I–593), `GraphicsExportSetInputHandle` (I–597), `GraphicsExportSetInputPtr` (I–602), `GraphicsExportSetInputGraphicsImporter` (I–595), `GraphicsExportSetInputPicture` (I–599), `GraphicsExportSetInputGWorld` (I–596), and `GraphicsExportSetInputPixmap` (I–600).

GraphicsExportGetInputGraphicsImporter

Returns the current input graphics importer instance for a graphics export operation.

```
ComponentResult GraphicsExportGetInputGraphicsImporter (
    GraphicsExportComponent    ci,
    GraphicsImportComponent    *grip );
```

ci

The component instance that identifies your connection to the graphics exporter component.

grip

Points to a variable to receive the source graphics importer.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You must get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a `QuickDrawPicture`, a graphics world, a `Pixmap` (IV–2332) structure, or a piece of compressed data described by an `ImageDescription` (IV–2285) structure. Compressed data can be in a

file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding set function, see `GraphicsExportSetInputGraphicsImporter` (I–595). The following other set functions also specify the sources for input images: `GraphicsExportSetInputDataReference` (I–593), `GraphicsExportSetInputFile` (I–594), `GraphicsExportSetInputHandle` (I–597), `GraphicsExportSetInputPtr` (I–602), `GraphicsExportSetInputPicture` (I–599), `GraphicsExportSetInputGWorld` (I–596), and `GraphicsExportSetInputPixmap` (I–600).

GraphicsExportGetInputGWorld

Returns the current input graphics world for a graphics export operation.

```
ComponentResult GraphicsExportGetInputGWorld (
    GraphicsExportComponent    ci,
    GWorldPtr                  *gworld );
```

ci

The component instance that identifies your connection to the graphics exporter component.

gworld

Points to a variable to receive the source graphics world.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a `QuickDrawPicture`, a graphics world, a `Pixmap` (IV–2332) structure, or a piece of compressed data described by an `ImageDescription` (IV–2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure

GraphicsExportGetInputHandle

that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V-2887)

See Also

For the corresponding set function, see `GraphicsExportSetInputGWorld` (I-596). The following other set functions also specify the sources for input images: `GraphicsExportSetInputDataReference` (I-593), `GraphicsExportSetInputFile` (I-594), `GraphicsExportSetInputHandle` (I-597), `GraphicsExportSetInputPtr` (I-602), `GraphicsExportSetInputGraphicsImporter` (I-595), `GraphicsExportSetInputPicture` (I-599), and `GraphicsExportSetInputPixmap` (I-600).

GraphicsExportGetInputHandle

Returns the current input handle for a graphics export operation.

```
ComponentResult GraphicsExportGetInputHandle (
    GraphicsExportComponent    ci,
    Handle                     *h );
```

ci

The component instance that identifies your connection to the graphics exporter component.

h

A pointer to receive the handle.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error. If the current source is not a handle, the function returns `paramErr`.

Discussion

You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a `QuickDrawPicture`, a graphics world, a `PixelFormat` (IV-2332) structure, or a piece of compressed data described by an `ImageDescription` (IV-2285) structure. Compressed data can be

in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding set function, see `GraphicsExportSetInputHandle` (I–597). The following other set functions also specify the sources for input images:

`GraphicsExportSetInputDataReference` (I–593), `GraphicsExportSetInputFile` (I–594), `GraphicsExportSetInputPtr` (I–602), `GraphicsExportSetInputGraphicsImporter` (I–595), `GraphicsExportSetInputPicture` (I–599), `GraphicsExportSetInputGWorld` (I–596), and `GraphicsExportSetInputPixmap` (I–600).

GraphicsExportGetInputImageDepth

Returns the depth of the input image for a graphics export operation.

```
ComponentResult GraphicsExportGetInputImageDepth (
    GraphicsExportComponent  ci,
    long                     *inputDepth );
```

ci

The component instance that identifies your connection to the graphics exporter component.

inputDepth

Points to a variable to receive the input image depth.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing a Graphics Exporter’s Input Image” (V–2889)

GraphicsExportGetInputImageDescription

See Also

When doing a standalone export, an exporter will typically call either `GraphicsExportGetInputImageDescription` (I–570), or both `GraphicsExportGetInputImageDimensions` (I–571) and `GraphicsExportGetInputImageDepth`, to determine the image’s bounds and depth, and then allocate an offscreen graphics world and call `GraphicsExportDrawInputImage` (I–557) to draw portions of the image into the graphics world.

GraphicsExportGetInputImageDescription

Returns an image description describing the input image in a graphics export operation.

```
ComponentResult GraphicsExportGetInputImageDescription (
    GraphicsExportComponent    ci,
    ImageDescriptionHandle     *desc );
```

ci

The component instance that identifies your connection to the graphics exporter component.

desc

Points to a variable to receive a handle to an `ImageDescription` (IV–2285) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns an `ImageDescription` (IV–2285) structure containing information such as the format of the compressed data, its bit depth, natural bounds, and resolution.

Special Considerations

The caller is responsible for disposing of the returned image description handle.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing a Graphics Exporter’s Input Image” (V–2889)

See Also

When doing a standalone export, an exporter will typically call `GraphicsExportGetInputImageDescription`, or `GraphicsExportGetInputImageDimensions` (I–571) and `GraphicsExportGetInputImageDepth` (I–569), to determine the image’s bounds and depth, and then allocate an offscreen graphics world and call `GraphicsExportDrawInputImage` (I–557) to draw portions of the image into the graphics world.

GraphicsExportGetInputImageDimensions

Returns the dimensions of the input image in a graphics export operation.

```
ComponentResult GraphicsExportGetInputImageDimensions (
    GraphicsExportComponent ci,
    Rect *dimensions );
```

ci

The component instance that identifies your connection to the graphics exporter component.

dimensions

Points to a rectangle to receive the dimensions of the input image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing a Graphics Exporter’s Input Image” (V–2889)

See Also

When doing a standalone export, an exporter will typically call `GraphicsExportGetInputImageDescription` (I–570), or `GraphicsExportGetInputImageDimensions` and `GraphicsExportGetInputImageDepth` (I–569), to determine the image’s bounds and depth, and then allocate an offscreen graphics world and call `GraphicsExportDrawInputImage` (I–557) to draw portions of the image into the graphics world.

GraphicsExportGetInputOffsetAndLimit

GraphicsExportGetInputOffsetAndLimit

Retrieves the current input offset and limit in a graphics export operation.

```
ComponentResult GraphicsExportGetInputOffsetAndLimit (
    GraphicsExportComponent    ci,
    unsigned long               *offset,
    unsigned long               *limit );
```

ci

The component instance that identifies your connection to the graphics exporter component.

offset

Points to a variable to receive the offset. If you don't need this information, pass NIL.

limit

Points to a variable to receive the limit. If you don't need this information, pass NIL.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function is only applicable when the input is a data reference, file, handle or pointer.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Restricting the Range of an Input Image’s Source” (V–2888)

See Also

For the corresponding set function, see GraphicsExportSetInputOffsetAndLimit (I–598).

GraphicsExportGetInputPicture

Returns the current input picture in a graphics export operation.

```
ComponentResult GraphicsExportGetInputPicture (
    GraphicsExportComponent    ci,
    PicHandle                  *picture );
```

ci

The component instance that identifies your connection to the graphics exporter component.

picture

Points to a variable to receive the source picture.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a graphics world, a `Pixmap` (IV–2332) structure, or a piece of compressed data described by an `ImageDescription` (IV–2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding set function, see `GraphicsExportSetInputPicture` (I–599). The following other set functions also specify the sources for input images: `GraphicsExportSetInputDataReference` (I–593), `GraphicsExportSetInputFile` (I–594), `GraphicsExportSetInputHandle` (I–597), `GraphicsExportSetInputPtr` (I–602), `GraphicsExportSetInputGraphicsImporter` (I–595), `GraphicsExportSetInputGWorld` (I–596), and `GraphicsExportSetInputPixmap` (I–600).

GraphicsExportGetInputPixmap

Returns the current input pixmap in a graphics export operation.

```
ComponentResult GraphicsExportGetInputPixmap (
    GraphicsExportComponent    ci,
    PixmapHandle                *pixmap );
```

GraphicsExportGetInputPtr

ci

The component instance that identifies your connection to the graphics exporter component.

pixmap

Points to a variable to receive the source `PixelFormat` (IV–2332) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a graphics world, a `PixelFormat` (IV–2332) structure, or a piece of compressed data described by an `ImageDescription` (IV–2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding set function, see `GraphicsExportSetInputPixelFormat` (I–600). The following other set functions also specify the sources for input images:

`GraphicsExportSetInputDataReference` (I–593), `GraphicsExportSetInputFile` (I–594), `GraphicsExportSetInputHandle` (I–597), `GraphicsExportSetInputPtr` (I–602), `GraphicsExportSetInputGraphicsImporter` (I–595), `GraphicsExportSetInputPicture` (I–599), and `GraphicsExportSetInputGWorld` (I–596).

GraphicsExportGetInputPtr

Returns the current input pointer in a graphics export operation.

```
ComponentResult GraphicsExportGetInputPtr (
    GraphicsExportComponent    ci,
    Ptr                        *p,
    unsigned long               *size );
```

ci

The component instance that identifies your connection to the graphics exporter component.

p

A pointer to receive a pointer containing the graphics data.

size

A pointer to a value describing the size of the image data in bytes.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a graphics world, a PixMap (IV–2332) structure, or a piece of compressed data described by an ImageDescription (IV–2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding set function, see GraphicsExportSetInputPtr (I–602). The following other set functions also specify the sources for input images: GraphicsExportSetInputDataReference (I–593), GraphicsExportSetInputFile (I–594), GraphicsExportSetInputHandle (I–597), GraphicsExportSetInputGraphicsImporter (I–595), GraphicsExportSetInputPicture (I–599), GraphicsExportSetInputGWorld (I–596), and GraphicsExportSetInputPixmap (I–600).

GraphicsExportGetInterlaceStyle

Returns the **interlace** style in a graphics export operation.

```
ComponentResult GraphicsExportGetInterlaceStyle (
    GraphicsExportComponent ci,
```

GraphicsExportGetMetaData

```
unsigned long                *interlaceStyle );
```

ci

The component instance that identifies your connection to the graphics exporter component.

interlaceStyle

Points to a variable to receive the **interlace** style. Valid values and interpretations are defined by individual exporters. In QuickTime 4, the PNG graphics exporter supports the `interlaceStyle` settings shown below

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

interlaceStyle Constants

`kQTPNGInterlaceNone`

No **interlace**. Value is 0.

`kQTPNGInterlaceAdam7`

Uses the 2D Adam7 algorithm. Value is 1.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding set function, see `GraphicsExportSetInterlaceStyle` (I–603).

GraphicsExportGetMetaData

Returns the current user data setting in a graphics export operation.

```
ComponentResult GraphicsExportGetMetaData (
    GraphicsExportComponent    ci,
    void                        *userData );
```

ci

The component instance that identifies your connection to the graphics exporter component.

userData

A pointer to a `UserDataRecord` (IV–2496) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4. In QuickTime 4, none of the supplied graphics exporters support setting user data.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding set function, see `GraphicsExportSetMetaData` (I–604).

GraphicsExportGetMIMETYPEList

Returns MIME types and other information about the graphics format in a graphics export operation.

```
ComponentResult GraphicsExportGetMIMETYPEList (
    GraphicsExportComponent  ci,
    void                    *qtAtomContainerPtr );
```

ci

The component instance that identifies your connection to the graphics exporter component.

qtAtomContainerPtr

Receives a newly-created QuickTime atom container that contains information about the graphics format.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function creates and returns a QuickTime atom container that contains the format’s name, as a string in an atom of type ‘desc’ (`kMIMEInfoDescriptionTag`), and optionally the MIME type as a string in an atom of type ‘mime’ (`kMIMEInfoMimeTypeTag`).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Finding Out About Graphics Export Image Formats” (V–2884)

GraphicsExportGetOutputDataReference

GraphicsExportGetOutputDataReference

Gets the output data reference handle in a graphics export operation.

```
ComponentResult GraphicsExportGetOutputDataReference (  
    GraphicsExportComponent    ci,  
    Handle                     *dataRef,  
    OSType                     *dataRefType );
```

ci

The component instance that identifies your connection to the graphics exporter component.

dataRef

Points to a variable to receive the data reference handle.

dataRefType

Points to a variable to receive a constant that identifies the data reference type. See “Data References” (IV–2661).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

The caller is responsible for disposing of the returned data reference handle.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Destinations for Output Images” (V–2890)

See Also

For the corresponding set function, see `GraphicsExportSetOutputDataReference` (I–604).

GraphicsExportGetOutputFile

Returns the current output file for a graphics export operation.

```
ComponentResult GraphicsExportGetOutputFile (  
    GraphicsExportComponent    ci,  
    FSSpec                     *theFile );
```


ci

The component instance that identifies your connection to the graphics exporter component.

theFile

Points to a variable to receive the FSSpec (IV–2262).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Destinations for Output Images” (V–2890)

See Also

For the corresponding set function, see `GraphicsExportSetOutputFile` (I–605).

GraphicsExportGetOutputFileTypeAndCreator

Gets the type and creator codes for the output file in a graphics export operation.

```
ComponentResult GraphicsExportGetOutputFileTypeAndCreator (
    GraphicsExportComponent  ci,
    OSType                  *fileType,
    OSType                  *fileCreator );
```

ci

The component instance that identifies your connection to the graphics exporter component.

fileType

Receives the file type for the new image file. See “File Types and Creators” (IV–2668).

fileCreator

Receives the file creator for the new image file. See “File Types and Creators” (IV–2668).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

GraphicsExportGetOutputHandle

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Destinations for Output Images” (V–2890)

See Also

For the corresponding set function, see

`GraphicsExportSetOutputFileTypeAndCreator` (I–606).

GraphicsExportGetOutputHandle

Returns the current output handle for a graphics export operation.

```
ComponentResult GraphicsExportGetOutputHandle (  
    GraphicsExportComponent    ci,  
    Handle                     *h );
```

ci

The component instance that identifies your connection to the graphics exporter component.

h

Points to a variable to receive the handle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Destinations for Output Images” (V–2890)

See Also

For the corresponding set function, see `GraphicsExportSetOutputHandle` (I–606).

GraphicsExportGetOutputMark

Returns the current file position for a graphics export operation.

```
ComponentResult GraphicsExportGetOutputMark (  
    GraphicsExportComponent    ci,  
    unsigned long              *mark );
```

ci

The component instance that identifies your connection to the graphics exporter component.

mark

Receives the current file position, as a byte offset from the beginning of the output data reference.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

Not all output data types support the current file position feature.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Writing Graphics Exporter Output Data” (V–2891)

See Also

For the corresponding set function, see `GraphicsExportSetOutputMark` (I–607).

GraphicsExportGetOutputOffsetAndMaxSize

Returns the output starting offset and maximum size limit for a graphics export operation.

```
ComponentResult GraphicsExportGetOutputOffsetAndMaxSize (
    GraphicsExportComponent    ci,
    unsigned long               *offset,
    unsigned long               *maxSize,
    Boolean                     *truncateFile );
```

ci

The component instance that identifies your connection to the graphics exporter component.

offset

On return, a value describing the byte offset of the image data from the beginning of the data reference. If you are not interested in this information, you may pass `NIL`.

maxSize

On return, a value describing the maximum size limit. If you are not interested in this information, you may pass `NIL`.

GraphicsExportGetProgressProc

`truncateFile`

A Boolean value; TRUE means to truncate the file, FALSE means not.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Destinations for Output Images” (V–2890)

See Also

For the corresponding set function, see `GraphicsExportSetOutputOffsetAndMaxSize` (I–608).

GraphicsExportGetProgressProc

Returns the current **progress function** for a graphics export operation.

```
ComponentResult GraphicsExportGetProgressProc (  
    GraphicsExportComponent    ci,  
    ICMProgressProcRecordPtr   progressProc );
```

`ci`

The component instance that identifies your connection to the graphics exporter component.

`progressProc`

A pointer to an `ICMProgressProc` (III–2093) callback.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

By default, graphics export components have no **progress functions**.

Special Considerations

This function is always implemented by the **base graphics exporter**.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting and Setting Progress Procs” (V–2887)

See Also

For the corresponding set function, see `GraphicsExportSetProgressProc` (I–608).

GraphicsExportGetResolution

Determines the resolution of a graphics exporter component.

```
ComponentResult GraphicsExportGetResolution (
    GraphicsExportComponent  ci,
    Fixed                    *horizontalResolution,
    Fixed                    *verticalResolution );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

horizontalResolution

Points to a variable to receive the horizontal resolution.

verticalResolution

Points to a variable to receive the vertical resolution.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding set function, see `GraphicsExportSetResolution` (I–609).

GraphicsExportGetSettingsAsAtomContainer

Retrieves the current settings from a graphics exporter component.

```
ComponentResult GraphicsExportGetSettingsAsAtomContainer (
    GraphicsExportComponent  ci,
    void                    *qtAtomContainerPtr );
```

GraphicsExportGetSettingsAsText

ci

The component instance that identifies your connection to the graphics exporter component.

qtAtomContainerPtr

Points to a variable to receive a new QuickTime atom container containing the current graphics exporter component settings.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

The caller is responsible for disposing of the returned atom container.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Obtaining Graphics Exporter Settings” (V–2885)

See Also

For the corresponding set function, see `GraphicsExportSetSettingsFromAtomContainer` (I–610).

GraphicsExportGetSettingsAsText

Retrieves the current settings from the graphics export component in a user-readable format.

```
ComponentResult GraphicsExportGetSettingsAsText (
    GraphicsExportComponent  ci,
    Handle                   *theText );
```

ci

The component instance that identifies your connection to the graphics exporter component.

theText

Points to a variable to receive a newly-allocated handle containing text.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

The caller is responsible for disposing of the returned handle.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Obtaining Graphics Exporter Settings” (V–2885)

GraphicsExportGetTargetDataSize

Returns the current desired maximum data size for a graphics export operation.

```
ComponentResult GraphicsExportGetTargetDataSize (  
    GraphicsExportComponent    ci,  
    unsigned long              *targetDataSize );
```

ci

The component instance that identifies your connection to the graphics exporter component.

targetDataSize

Points to a variable to receive the desired maximum data size in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding set function, see `GraphicsExportSetTargetDataSize` (I–611).

GraphicsExportMayExporterReadInputData

Asks whether the image source for a graphics export operation is in a form that the exporter can read.

```
ComponentResult GraphicsExportMayExporterReadInputData (  
    GraphicsExportComponent    ci,  
    Boolean                   *mayReadInputData );
```

ci

The component instance that identifies your connection to the graphics exporter component.

GraphicsExportReadInputData

`mayReadInputData`

Points to a Boolean; TRUE means that the image source is in a form that the exporter can read, FALSE means it is not.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Some kinds of image source, such as files and handles, form a stream of bytes that can be read directly. Others, such as pictures and pixmaps, do not. Format-specific graphics exporters usually cannot transcode if they cannot read the original data, so those exporters which implement `GraphicsExportCanTranscode` (I–552) will usually first call `GraphicsExportMayExporterReadInputData`.

Special Considerations

This function is used by format-specific graphics exporters when transcoding. Applications will not normally need to call this function.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Reading Graphics Exporter Input Data” (V–2889)

GraphicsExportReadInputData

Reads the original image data in a graphics export operation.

```
ComponentResult GraphicsExportReadInputData (
    GraphicsExportComponent    ci,
    void                        *dataPtr,
    unsigned long               dataOffset,
    unsigned long               dataSize );
```

`ci`

The component instance that identifies your connection to the graphics exporter component.

`dataPtr`

A pointer to a memory block to receive the data.

`dataOffset`

The offset of the image data within the source image data. The function begins reading image data from this offset.

`dataSize`

The number of bytes of image data to read.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function communicates with the appropriate data handler to retrieve image data.

Special Considerations

This function is used by format-specific graphics exporters when transcoding. Applications will not normally need to call this function.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Reading Graphics Exporter Input Data” (V–2889)

GraphicsExportReadOutputData

Reads output image data in a graphics export operation.

```
ComponentResult GraphicsExportReadOutputData (  
    GraphicsExportComponent    ci,  
    void                       *dataPtr,  
    unsigned long               dataOffset,  
    unsigned long               dataSize );
```

`ci`

The component instance that identifies your connection to the graphics exporter component.

`dataPtr`

A pointer to a memory block to receive the data.

`dataOffset`

The offset of the image data within the data reference. The function begins reading image data from this offset.

`dataSize`

The number of bytes of image data to read.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

GraphicsExportRequestSettings

Special Considerations

Not all output data types support this function.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Writing Graphics Exporter Output Data” (V–2891)

GraphicsExportRequestSettings

Displays a dialog for the user to configure graphics exporter settings, if applicable.

```
ComponentResult GraphicsExportRequestSettings (
    GraphicsExportComponent    ci,
    ModalFilterYDUPP          filterProc,
    void                      *yourDataProc );
```

ci

The component instance that identifies your connection to the graphics exporter component.

filterProc

A `ModalFilterYDProc` (III–2099) callback. If you don’t need one, pass `NIL`.

yourDataProc

An extra parameter that will be passed to the `ModalFilterProc` callback when it is called. If you don’t need one, pass `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Some graphics exporters don’t support settings dialogs, and so don’t implement this call. To find out whether a graphics exporter implements this call, you can use this code:

```
CallComponentCanDo( myGraphicsExporter,
                    kGraphicsExportRequestSettingsSelect);
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Obtaining Graphics Exporter Settings” (V–2885)

See Also

See `CallComponentCanDo` (I–58).

GraphicsExportSetColorSyncProfile

Sets the ColorSync profile to embed in the image file for a graphics export operation.

```
ComponentResult GraphicsExportSetColorSyncProfile (
    GraphicsExportComponent  ci,
    Handle                   colorSyncProfile );
```

ci

The component instance that identifies your connection to the graphics exporter component.

colorSyncProfile

A handle to the ColorSync profile.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

ColorSync profiles allow image files to describe their native colorspace in a self-contained manner. They can be stored in atoms of type ‘`iccc`’ (IV–2547).

Version Notes

Introduced in QuickTime 4. Starting with QuickTime 4, the JPEG, PNG, PICT, QuickTime Image and TIFF graphics exporters support embedded ColorSync profiles.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding get function, see `GraphicsExportGetColorSyncProfile` (I–558).

GraphicsExportSetCompressionMethod

Defines the compression method to use in a graphics export operation.

```
ComponentResult GraphicsExportSetCompressionMethod (
    GraphicsExportComponent  ci,
    long                     compressionMethod );
```

GraphicsExportSetCompressionQuality

ci

The component instance that identifies your connection to the graphics exporter component.

compressionMethod

A value (see below) describing the compression algorithm to be used by the graphics exporter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

compressionMethod Constants

`kQTTIFFCompression_None`

No compression. This value is 1.

`kQTTIFFCompression_PackBits`

PackBits compression. This value is 32773L

Discussion

In QuickTime 4, the TIFF graphics exporter supports the `compressionMethod` settings `kQTTIFFCompression_None` and `kQTTIFFCompression_PackBits`. Some image formats, such as TIFF, support several compression methods.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding get function, see `GraphicsExportGetCompressionMethod` (I–559).

GraphicsExportSetCompressionQuality

Defines the compression quality for a graphics export operation.

```
ComponentResult GraphicsExportSetCompressionQuality (
    GraphicsExportComponent  ci,
    CodecQ                   spatialQuality );
```

ci

The component instance that identifies your connection to the graphics exporter component.

`spatialQuality`

A constant (see below) that defines the currently specified quality value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

spatialQuality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

This setting is only supported by lossy compression methods.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding get function, see `GraphicsExportGetCompressionQuality` (I–559).

GraphicsExportSetDepth

Defines the depth to use in a graphics export operation.

```
ComponentResult GraphicsExportSetDepth (
    GraphicsExportComponent ci,
    long depth );
```

GraphicsExportSetDontRecompress

ci

The component instance that identifies your connection to the graphics exporter component.

depth

A value describing the depth of the image data. Some image file formats support more than one pixel depth.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The BMP, JPEG, Photoshop, PNG, PICT, QuickTime Image, TGA and TIFF graphics exporters support the depth setting. Some image file formats support more than one pixel depth.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding get function, see `GraphicsExportGetDepth` (I–562).

GraphicsExportSetDontRecompress

Requests that the original compressed data for a graphics export operation not be decompressed and recompressed, but be copied through to the output file.

```
ComponentResult GraphicsExportSetDontRecompress (
    GraphicsExportComponent    ci,
    Boolean                    dontRecompress );
```

ci

The component instance that identifies your connection to the graphics exporter component.

dontRecompress

If `TRUE`, requests not to recompress the image data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Even though it is not decompressed and recompressed, graphics data may be modified when it is copied through.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding get function, see `GraphicsExportGetDontRecompress` (I–562).

To determine whether a particular export setting is available, use

`CallComponentCanDo` (I–58).

GraphicsExportSetInputDataReference

Specifies that the source image for a graphics export operation is a compressed image stored in a data reference.

```
ComponentResult GraphicsExportSetInputDataReference (
    GraphicsExportComponent  ci,
    Handle                   dataRef,
    OSType                   dataRefType,
    ImageDescriptionHandle    desc );
```

ci

The component instance that identifies your connection to the graphics exporter component.

dataRef

A QuickTime data reference. See “Data References” (IV–2661).

dataRefType

The type of the data reference; see “Data References” (IV–2661).

desc

A handle to an `ImageDescription` (IV–2285) structure, describing the compressed data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (I–554). The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a graphics world, a `PixMap` (IV–2332) structure, or a piece of compressed data described by an `ImageDescription` (IV–2285) structure.

Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics

GraphicsExportSetInputFile

exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding get function, see `GraphicsExportGetInputDataReference` (I–563). The following other functions let you get the sources for input images: `GraphicsExportGetInputFile` (I–565), `GraphicsExportGetInputHandle` (I–568), `GraphicsExportGetInputPtr` (I–574), `GraphicsExportGetInputGraphicsImporter` (I–566), `GraphicsExportGetInputPicture` (I–572), `GraphicsExportGetInputGWorld` (I–567), and `GraphicsExportGetInputPixmap` (I–573).

GraphicsExportSetInputFile

Specifies that the source image for a graphics export operation is a compressed image stored in a file.

```
ComponentResult GraphicsExportSetInputFile (
    GraphicsExportComponent    ci,
    const FSSpec               *theFile,
    ImageDescriptionHandle      desc );
```

ci

The component instance that identifies your connection to the graphics exporter component.

theFile

A pointer to the `FSSpec` (IV–2262) structure for the file containing the graphics data.

desc

A handle to an `ImageDescription` (IV–2285) structure that describes the compressed data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (I-554). The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a graphics world, a `PixMap` (IV-2332) structure, or a piece of compressed data described by an `ImageDescription` (IV-2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V-2887)

See Also

For the corresponding get function, see `GraphicsExportGetInputFile` (I-565). The following other functions let you get the sources for input images: `GraphicsExportGetInputDataReference` (I-563), `GraphicsExportGetInputHandle` (I-568), `GraphicsExportGetInputPtr` (I-574), `GraphicsExportGetInputGraphicsImporter` (I-566), `GraphicsExportGetInputPicture` (I-572), `GraphicsExportGetInputGWorld` (I-567), and `GraphicsExportGetInputPixmap` (I-573).

GraphicsExportSetInputGraphicsImporter

Specifies that the source image for a graphics export operation is to be drawn by a graphics importer instance.

```
ComponentResult GraphicsExportSetInputGraphicsImporter (
    GraphicsExportComponent    ci,
    GraphicsImporterComponent  grip );
```

`ci`

The component instance that identifies your connection to the graphics exporter component.

`grip`

The source graphics importer component instance.

GraphicsExportSetInputGWorld

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

You can use this function to specify a source before you call GraphicsExportDoExport (I–554). The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a graphics world, a PixMap (IV–2332) structure, or a piece of compressed data described by an ImageDescription (IV–2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Special Considerations

It is the caller’s responsibility to dispose of the graphics importer.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding get function, see GraphicsExportGetInputGraphicsImporter (I–566). The following other functions let you get the sources for input images: GraphicsExportGetInputDataReference (I–563), GraphicsExportGetInputFile (I–565), GraphicsExportGetInputHandle (I–568), GraphicsExportGetInputPtr (I–574), GraphicsExportGetInputPicture (I–572), GraphicsExportGetInputGWorld (I–567), and GraphicsExportGetInputPixmap (I–573).

GraphicsExportSetInputGWorld

Specifies that the source image for a graphics export operation is a graphics world.

```
ComponentResult GraphicsExportSetInputGWorld (
    GraphicsExportComponent    ci,
    GWorldPtr                  gworld );
```

ci

The component instance that identifies your connection to the graphics exporter component.

`gworld`

The source graphics world. It must be a real graphics world; you may not pass an ordinary color `GrafPort`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (I–554). The source can be a QuickTime graphics importer component instance, a `QuickDrawPicture`, a graphics world, a `PixelFormat` (IV–2332) structure, or a piece of compressed data described by an `ImageDescription` (IV–2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Special Considerations

The graphics exporter will never dispose the graphics world.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding get function, see `GraphicsExportGetInputGWorld` (I–567). The following other functions let you get the sources for input images: `GraphicsExportGetInputDataReference` (I–563), `GraphicsExportGetInputFile` (I–565), `GraphicsExportGetInputHandle` (I–568), `GraphicsExportGetInputPtr` (I–574), `GraphicsExportGetInputGraphicsImporter` (I–566), `GraphicsExportGetInputPicture` (I–572), and `GraphicsExportGetInputPixelFormat` (I–573).

GraphicsExportSetInputHandle

Specifies that the source image for a graphics export operation is a compressed image referenced by a handle.

```
ComponentResult GraphicsExportSetInputHandle (
    GraphicsExportComponent    ci,
    Handle                    h,
    ImageDescriptionHandle     desc );
```

GraphicsExportSetInputOffsetAndLimit

ci

The component instance that identifies your connection to the graphics exporter component.

h

A handle to graphics data.

desc

A handle to an `ImageDescription` (IV–2285) structure that describes the compressed data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (I–554). The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a graphics world, a `PixMap` (IV–2332) structure, or a piece of compressed data described by an `ImageDescription` (IV–2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding get function, see `GraphicsExportGetInputHandle` (I–568). The following other functions let you get the sources for input images: `GraphicsExportGetInputDataReference` (I–563), `GraphicsExportGetInputFile` (I–565), `GraphicsExportGetInputPtr` (I–574), `GraphicsExportGetInputGraphicsImporter` (I–566), `GraphicsExportGetInputPicture` (I–572), `GraphicsExportGetInputGWorld` (I–567), and `GraphicsExportGetInputPixmap` (I–573).

GraphicsExportSetInputOffsetAndLimit

Specifies the portion of an input data reference, file, handle or pointer that a graphics exporter is permitted to read.

```
ComponentResult GraphicsExportSetInputOffsetAndLimit (
    GraphicsExportComponent    ci,
    unsigned long              offset,
    unsigned long              limit );
```

ci

The component instance that identifies your connection to the graphics exporter component.

offset

The byte offset of the input image data from the beginning of the data reference.

limit

The offset of the byte following the last byte of the input image data. (If you don't need to apply any limit, pass `(unsigned long)-1`.) Both the *offset* parameter and the *limit* parameter values are relative to the start of the compressed data. `GraphicsExportGetInputDataSize` (I-564) and `GraphicsExportReadInputData` (I-586) take the offset and limit values into account automatically.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This routine would be useful if, for example, the source was a JPEG image embedded within a larger file.

Special Considerations

This function is only applicable when the input is a data reference, file, handle, or pointer.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Restricting the Range of an Input Image’s Source” (V-2888)

See Also

For the corresponding get function, see `GraphicsExportGetInputOffsetAndLimit` (I-572).

GraphicsExportSetInputPicture

Specifies that the source image for a graphics export operation is a picture.

GraphicsExportSetInputPixmap

```
ComponentResult GraphicsExportSetInputPicture (
    GraphicsExportComponent    ci,
    PicHandle                   picture );
```

ci

The component instance that identifies your connection to the graphics exporter component.

picture

A handle to the source picture.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

You can use this function to specify a source before you call GraphicsExportDoExport (I–554). The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a graphics world, a Pixmap (IV–2332) structure, or a piece of compressed data described by an ImageDescription (IV–2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding get function, see GraphicsExportGetInputPicture (I–572). The following other functions let you get the sources for input images: GraphicsExportGetInputDataReference (I–563), GraphicsExportGetInputFile (I–565), GraphicsExportGetInputHandle (I–568), GraphicsExportGetInputPtr (I–574), GraphicsExportGetInputGraphicsImporter (I–566), GraphicsExportGetInputGWorld (I–567), and GraphicsExportGetInputPixmap (I–573).

GraphicsExportSetInputPixmap

Specifies that the source image for a graphics export operation is a pixmap.

```
ComponentResult GraphicsExportSetInputPixmap (
```

```
GraphicsExportComponent    ci,  
PixmapHandle               pixmap );
```

ci

The component instance that identifies your connection to the graphics exporter component.

pixmap

The source `Pixmap` (IV-2332) structure.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (I-554). The source can be a QuickTime graphics importer component instance, a `QuickDrawPicture`, a graphics world, a `Pixmap` (IV-2332) structure, or a piece of compressed data described by an `ImageDescription` (IV-2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Special Considerations

It is the caller’s responsibility to dispose of the `pixmap`.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V-2887)

See Also

For the corresponding get function, see `GraphicsExportGetInputPixmap` (I-573). The following other functions let you get the sources for input images: `GraphicsExportGetInputDataReference` (I-563), `GraphicsExportGetInputFile` (I-565), `GraphicsExportGetInputHandle` (I-568), `GraphicsExportGetInputPtr` (I-574), `GraphicsExportGetInputGraphicsImporter` (I-566), `GraphicsExportGetInputPicture` (I-572), and `GraphicsExportGetInputGWorld` (I-567).

GraphicsExportSetInputPtr

Specifies that the source image for a graphics export operation is a compressed image stored at a fixed address in memory.

```
ComponentResult GraphicsExportSetInputPtr (  
    GraphicsExportComponent    ci,  
    Ptr                        p,  
    unsigned long              size,  
    ImageDescriptionHandle     desc );
```

ci

The component instance that identifies your connection to the graphics exporter component.

p

A pointer to a value the image.

size

A value describing the size of the image data in bytes.

desc

A handle to an `ImageDescription` (IV–2285) structure that describes the compressed data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (I–554). The source can be a QuickTime graphics importer component instance, a QuickDraw Picture, a graphics world, a `Pixmap` (IV–2332) structure, or a piece of compressed data described by an `ImageDescription` (IV–2285) structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the **base graphics exporter**; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Sources for Graphics Exporter Input Images” (V–2887)

See Also

For the corresponding get function, see `GraphicsExportGetInputPtr` (I–574). The following other functions let you get the sources for input images: `GraphicsExportGetInputDataReference` (I–563), `GraphicsExportGetInputFile` (I–565), `GraphicsExportGetInputHandle` (I–568), `GraphicsExportGetInputGraphicsImporter` (I–566), `GraphicsExportGetInputPicture` (I–572), `GraphicsExportGetInputGWorld` (I–567), and `GraphicsExportGetInputPixmap` (I–573).

GraphicsExportSetInterlaceStyle

Defines the **interlace** style for a graphics export operation.

```
ComponentResult GraphicsExportSetInterlaceStyle (
    GraphicsExportComponent    ci,
    unsigned long               interlaceStyle );
```

ci

The component instance that identifies your connection to the graphics exporter component.

interlaceStyle

The new **interlace** style to use. Valid values and interpretations are defined by individual exporters. In QuickTime 4, the PNG graphics exporter supports the *interlaceStyle* settings shown below.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

interlaceStyle Constants

`kQTPNGInterlaceNone`

No **interlace**. Value is 0.

`kQTPNGInterlaceAdam7`

Uses the 2D Adam7 algorithm. Value is 1.

Discussion

A common use for this function is in the PNG and GIF formats, which rearrange data so that low-resolution images can be displayed from incomplete data streams.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

GraphicsExportSetMetaData

See Also

For the corresponding get function, see GraphicsExportGetInterlaceStyle (I–575).

GraphicsExportSetMetaData

Defines supplemental data for a graphics export operation, such as copyright text.

```
ComponentResult GraphicsExportSetMetaData (  
    GraphicsExportComponent    ci,  
    void                       *userData );
```

ci

The component instance that identifies your connection to the graphics exporter component.

userData

A pointer to user data. The value you pass should have the type *userData*, which is a pointer to a *UserDataRecord* (IV–2496).

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Special Considerations

In QuickTime 4, none of the supplied graphics exporters support setting user data.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: *ImageCompression.h*

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding get function, see GraphicsExportGetMetaData (I–576).

GraphicsExportSetOutputDataReference

Returns the current output data reference for a graphics export operation.

```
ComponentResult GraphicsExportSetOutputDataReference (  
    GraphicsExportComponent    ci,  
    Handle                     dataRef,  
    OSType                     dataRefType );
```

ci

The component instance that identifies your connection to the graphics exporter component.

dataRef

A QuickTime data reference.

dataRefType

The type of the data reference; see “Data References” (IV–2661).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Destinations for Output Images” (V–2890)

See Also

For the corresponding get function, see `GraphicsExportGetOutputDataReference` (I–578).

GraphicsExportSetOutputFile

Defines the output file for a graphics export operation.

```
ComponentResult GraphicsExportSetOutputFile (
    GraphicsExportComponent ci,
    const FSSpec             *theFile );
```

ci

The component instance that identifies your connection to the graphics exporter component.

theFile

an `FSSpec` (IV–2262) structure that identifies the file.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying Destinations for Output Images” (V–2890)

GraphicsExportSetOutputFileTypeAndCreator

See Also

For the corresponding get function, see GraphicsExportGetOutputFile (I–578).

GraphicsExportSetOutputFileTypeAndCreator

Sets the file type and creator codes for the output file of a graphics export operation.

```
ComponentResult GraphicsExportSetOutputFileTypeAndCreator (
    GraphicsExportComponent    ci,
    OSType                    fileType,
    OSType                    fileCreator );
```

ci

The component instance that identifies your connection to the graphics exporter component.

fileType

The file type for the new image file, such as 'JPEG'. See “File Types and Creators” (IV–2668).

fileCreator

The file creator for the new image file. This parameter may be 0, in which case a default file creator for this file type is used. See “File Types and Creators” (IV–2668).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Specifying Destinations for Output Images” (V–2890)

See Also

For the corresponding get function, see GraphicsExportGetOutputFileTypeAndCreator (I–579).

GraphicsExportSetOutputHandle

Sets a handle to the output of a graphics export operation.

```
ComponentResult GraphicsExportSetOutputHandle (
    GraphicsExportComponent    ci,
```

```
Handle h );
```

ci

The component instance that identifies your connection to the graphics exporter component.

h

The output handle.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Specifying Destinations for Output Images” (V–2890)

See Also

For the corresponding get function, see GraphicsExportGetOutputHandle (I–580).

GraphicsExportSetOutputMark

Seeks to the specified file position in a graphics export operation.

```
ComponentResult GraphicsExportSetOutputMark (
    GraphicsExportComponent ci,
    unsigned long mark );
```

ci

The component instance that identifies your connection to the graphics exporter component.

mark

The new file position, specified as a byte offset from the beginning of the output data reference.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Writing Graphics Exporter Output Data” (V–2891)

GraphicsExportSetOutputOffsetAndMaxSize

See Also

For the corresponding get function, see GraphicsExportGetOutputMark (I–580).

GraphicsExportSetOutputOffsetAndMaxSize

Specifies the output starting offset and maximum size limit for a graphics export operation.

```
ComponentResult GraphicsExportSetOutputOffsetAndMaxSize (  
    GraphicsExportComponent    ci,  
    unsigned long              offset,  
    unsigned long              maxSize,  
    Boolean                    truncateFile );
```

ci

The component instance that identifies your connection to the graphics exporter component.

offset

The byte offset of the image data from the beginning of the data reference.

maxSize

A value describing the maximum size limit.

truncateFile

A Boolean value; TRUE means to truncate the file.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Specifying Destinations for Output Images” (V–2890)

See Also

For the corresponding get function, see GraphicsExportGetOutputOffsetAndMaxSize (I–581).

GraphicsExportSetProgressProc

Installs a **progress function** in a graphics export operation.

```
ComponentResult GraphicsExportSetProgressProc (
```

```
GraphicsExportComponent    ci,
ICMProgressProcRecordPtr  progressProc );
```

ci

The component instance that identifies your connection to the graphics exporter component.

progressProc

Points to an ICMProgressProc (III–2093) callback. If you pass a value of –1, QuickTime provides a standard **progress function**. If you want to remove the existing progress function, pass NIL.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function is always implemented by the **base graphics exporter**.

Special Considerations

If your **progress function** does any drawing, you should take care to set a safe graphics state before doing so, and to restore the graphics state afterwards. In particular, the current graphics device may be an offscreen device.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Getting and Setting Progress Procs” (V–2887)

See Also

For the corresponding get function, see GraphicsExportGetProgressProc (I–582).

GraphicsExportSetResolution

Defines the resolution to store in the image file for a graphics export operation.

```
ComponentResult GraphicsExportSetResolution (
    GraphicsExportComponent    ci,
    Fixed                      horizontalResolution,
    Fixed                      verticalResolution );
```

ci

The component instance that identifies your connection to the graphics exporter component.

GraphicsExportSetSettingsFromAtomContainer

horizontalResolution

A value describing the horizontal resolution of the image, where the upper byte is dots per inch. The value 0x00480000 represents 72.0 dpi.

verticalResolution

A value describing the vertical resolution of the image, where the upper byte is dots per inch. The value 0x00480000 represents 72.0 dpi.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

See Also

For the corresponding get function, see `GraphicsExportGetResolution` (I–583).

GraphicsExportSetSettingsFromAtomContainer

Sets the graphics exporter component’s current configuration to match the settings in a passed atom container.

```
ComponentResult GraphicsExportSetSettingsFromAtomContainer (
    GraphicsExportComponent    ci,
    void                        *qtAtomContainer );
```

ci

The component instance that identifies your connection to the graphics exporter component.

qtAtomContainer

A pointer to a QuickTime atom container that contains settings.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The settings atom container may contain atoms other than those expected by the graphics exporter component or may be missing certain atoms. This function will use only the settings it understands.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Obtaining Graphics Exporter Settings” (V–2885)

GraphicsExportSetTargetDataSize

Defines a desired maximum data size for a graphics export operation and asks for a quality that does not exceed that size.

```
ComponentResult GraphicsExportSetTargetDataSize (  
    GraphicsExportComponent    ci,  
    unsigned long               targetDataSize );
```

ci

The component instance that identifies your connection to the graphics exporter component.

targetDataSize

A value that describes the maximum size of the image data in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Accessing Graphics Exporter Settings” (V–2885)

GraphicsExportWriteOutputData

Writes output image data in a graphics export operation.

```
ComponentResult GraphicsExportWriteOutputData (  
    GraphicsExportComponent    ci,  
    const void                 *dataPtr,  
    unsigned long              dataSize );
```

ci

The component instance that identifies your connection to the graphics exporter component.

dataPtr

A pointer to a memory block containing the data.

GraphicsImageImportGetSequenceEnabled

`dataSize`

The number of bytes of image data to write.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is used by format-specific graphics exporters to write output data.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Writing Graphics Exporter Output Data” (V–2891)

GraphicsImageImportGetSequenceEnabled

Undocumented

```
ComponentResult GraphicsImageImportGetSequenceEnabled (
    GraphicImageMovieImportComponent    ci,
    Boolean                             *enable );
```

`ci`

The component instance that identifies your connection to the movie importer component.

`enable`

A pointer to a Boolean that returns `TRUE` if enabled, `FALSE` otherwise.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding set function, see `GraphicsImageImportSetSequenceEnabled` (I–612).

GraphicsImageImportSetSequenceEnabled

Undocumented

```
ComponentResult GraphicsImageImportSetSequenceEnabled (
    GraphicImageMovieImportComponent    ci,
    Boolean                             enable );
```

ci

The component instance that identifies your connection to the movie importer component.

enable

Pass TRUE to enable, FALSE to disable.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

See Also

For the corresponding get function, see GraphicsImageImportGetSequenceEnabled (I–612).

GraphicsImportDoesDrawAllPixels

Asks whether the graphics importer expects to draw every pixel.

```
ComponentResult GraphicsImportDoesDrawAllPixels (
    GraphicsImportComponent    ci,
    short                      *drawsAllPixels );
```

ci

The component instance that identifies your connection to the graphics importer component.

drawsAllPixels

A pointer to a value (see below) that describes the predicted drawing behavior.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

drawsAllPixels Constants

graphicsImporterDrawsAllPixels

Every pixel in the boundary rectangle will be drawn. Value is 0.

graphicsImporterDoesntDrawAllPixels

Some pixels in the boundary rectangle will not be drawn. Value is 1.

GraphicsImporterDoExportImageFileDialog

`graphicsImporterDontKnowIfDrawAllPixels`

The graphics importer cannot determine whether all pixels will be drawn.
Value is 2.

Discussion

Some image file formats permit non-rectangular images or images with transparent regions. When such an image is drawn, not every pixel in the boundary rectangle will be changed. `GraphicsImportDoesDrawAllPixels` lets you try to find out whether this will be the case. For instance, you might choose to erase the area behind the image before drawing. If the graphics import component supports this function, `drawsAllPixels` will contain one of the constants shown above on return.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting Image Characteristics” (V–2878)

Related Java Methods

`quicktime.std.image.GraphicsImporter.doesDrawAllPixels()`

See Also

Format-specific graphics importers always implement `GraphicsImportGetImageDescription` (I–638) and may optionally implement `GraphicsImportGetNaturalBounds` (I–642), `GraphicsImportGetDataOffsetAndSize` (I–624), `GraphicsImportValidate` (I–665), and `GraphicsImportGetMetaData` (I–640), as well as `GraphicsImportDoesDrawAllPixels`.

GraphicsImportDoExportImageFileDialog

Presents a dialog box letting the user save an imported image in a foreign file format.

```
ComponentResult GraphicsImportDoExportImageFileDialog (
    GraphicsImportComponent    ci,
    const FSSpec                *inDefaultSpec,
    StringPtr                   prompt,
    ModalFilterYDUPP            filterProc,
    OSType                      *outExportedType,
    FSSpec                      *outExportedSpec,
    ScriptCode                   *outScriptTag );
```

`ci`

The component instance that identifies your connection to the graphics importer component.

`inDefaultSpec`

A pointer to an `FSSpec` (IV-2262) that suggests a default name for the file. If you don't want to suggest a default name, pass `NIL`.

`prompt`

A pointer to a prompt string that appears in the standard put dialog box; it may be `NIL`, in which case a default string is used.

`filterProc`

A modal filter function to be passed to the Mac OS function `CustomPutFile`; see *Inside Macintosh: Files* (listed in the **bibliography**) for more information. If you don't need to filter events, pass `NIL`.

`outExportedType`

A pointer to a variable that will receive the type of the export file that was chosen by the user. If you don't want this information, pass `NIL`. See “File Types and Creators” (IV-2668).

`outExportedSpec`

A pointer to a variable that will receive the `FSSpec` (IV-2262) of the file that was written. If you don't want this information, pass `NIL`.

`outScriptTag`

A pointer to a variable that will receive the script system in which the exported file name is to be displayed. See “Localization Codes” (IV-2673). If you don't want this information, pass `NIL`.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This function presents the user with an extended Standard File dialog box that allows the image currently in use by the graphics import component to be exported to a file, in a format of the user's choice.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Saving Image Files” (V-2881)

Related Java Methods

`quicktime.io.QTFile.fromGraphicsImporter()`,

`quicktime.std.image.GraphicsImporter.doExportImageFileDialog()`

GraphicsImportDraw

GraphicsImportDraw

Draws an imported image.

```
ComponentResult GraphicsImportDraw (  
    GraphicsImportComponent    ci );
```

ci

The component instance that identifies your connection to the graphics importer component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function draws the image currently in use by the graphics import component to the graphics port and device specified by `GraphicsImportSetGWorld` (I–660). `GraphicsImportDraw` takes into account all settings previously specified for the image, such as the source rectangle, transformation matrix, clipping region, graphics mode, and image quality.

Special Considerations

The **base graphics importer**’s drawing function uses the results of `GraphicsImportGetImageDescription` (I–638) and `GraphicsImportGetDataOffsetAndSize` (I–624) to create a decompression sequence, which it uses to draw the image. Subsequent draw operations with the same connection may reuse the decompression sequence. Other graphics importers may override this behavior.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Drawing Imported Images” (V–2880)

Related Java Methods

```
quicktime.std.image.GraphicsImporter.draw()
```

GraphicsImportExportImageFile

Saves an imported image in a foreign file format.

```
ComponentResult GraphicsImportExportImageFile (  
    GraphicsImportComponent    ci,  
    OSType                    fileType,
```

```
OSType          fileCreator,
const FSSpec     *fss,
ScriptCode      scriptTag );
```

ci

The component instance that identifies your connection to the graphics importer component.

fileType

The file type for the new image file, such as 'JPEG'. See “File Types and Creators” (IV–2668).

fileCreator

The file creator for the new image file. See “File Types and Creators” (IV–2668). You may pass 0, in which case a default file creator for this file type is used.

fss

A pointer to the FSSpec (IV–2262) structure that identifies the file that is to receive the exported image.

scriptTag

The script system in which the file name is to be displayed; see “Localization Codes” (IV–2673). If you have established the name and location of the file using one of the **Standard File Package** functions, use the script code returned in the reply record (reply.sfScript). Otherwise, specify the system script by setting the scriptTag parameter to the value smSystemScript. See *Inside Macintosh: Files* (listed in the **bibliography**) for more information about script specifications.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function creates a new file containing the image currently in use by the graphics import component. The new file is compressed in a format corresponding to the provided file type. If a non-identity matrix has been applied to the graphics import component, this matrix is applied to the image before export. Since most image formats don't support nonzero top-left coordinates, the matrix is temporarily adjusted to ensure that the exported image's bounds have top-left coordinates at (0,0). If the matrix does not map the graphics import component's source rectangle to a rectangle, there will be extra white space left around the image.

Special Considerations

Graphics import components can save data in several formats, including QuickDraw pictures and QuickTime Image files. This capability is only needed by applications that perform file format translation. Applications that only wish to draw the image can use GraphicsImportDraw (I–616).

GraphicsImportGetAliasedDataReference

Version Notes

In QuickTime 3, the supported export file types are `kQTFileTypePicture`, `kQTFileTypeQuickTimeImage`, `kQTFileTypeBMP`, `kQTFileTypeJPEG`, and `kQTFileTypePhotoShop`. QuickTime 4 uses graphics exporter components to implement image export.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Saving Image Files” (V–2881)

Related Java Methods

`quicktime.std.image.GraphicsImporter.exportImageFile()`

GraphicsImportGetAliasedDataReference

Deprecated.

```
ComponentResult GraphicsImportGetAliasedDataReference (
    GraphicsImportComponent    ci,
    Handle                     *dataRef,
    OSType                     *dataRefType );
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime.

Programming Info

C interface file: `ImageCompression.h`

GraphicsImportGetAsPicture

Creates a QuickDraw picture handle to an imported image.

```
ComponentResult GraphicsImportGetAsPicture (
    GraphicsImportComponent    ci,
    PicHandle                  *picture );
```

`ci`

The component instance that identifies your connection to the graphics importer component.

`picture`

Points to a handle to a `Picture` (IV–2331) structure that is to receive the image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function creates a new QuickDraw picture handle containing the image currently in use by the graphics import component. If possible, the image will remain in the compressed format. For example, if the image is from a JFIF file, the picture will contain compressed JPEG data. It is the responsibility of the caller to dispose of the picture handle.

Special Considerations

Graphics import components can save data in several formats, including QuickDraw pictures and QuickTime Image files. This capability is only needed by applications that perform file format translation. Applications that only wish to draw the image can use `GraphicsImportDraw` (I–616).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Saving Image Files” (V–2881)

Related Java Methods

```
quicktime.qd.Pict.fromGraphicsImporter(),  
quicktime.std.image.GraphicsImporter.getAsPicture()
```

GraphicsImportGetBoundsRect

Returns the bounding rectangle for drawing an imported image.

```
ComponentResult GraphicsImportGetBoundsRect (  
    GraphicsImportComponent    ci,  
    Rect                      *bounds );
```

ci

The component instance that identifies your connection to the graphics importer component.

bounds

A pointer to a `Rect` (IV–2399) structure describing the bounding rectangle that has been defined for the image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

GraphicsImportGetClip

Discussion

This is a convenience function. It is implemented by calling `GraphicsImportGetMatrix` (I–639) and `GraphicsImportGetNaturalBounds` (I–642) and using the results to calculate the drawing rectangle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Setting Drawing Parameters” (V–2879)

Related Java Methods

`quicktime.std.image.GraphicsImporter.getBoundsRect()`

See Also

For the corresponding set function, see `GraphicsImportSetBoundsRect` (I–649).

GraphicsImportGetClip

Returns the current clipping region for an imported image.

```
ComponentResult GraphicsImportGetClip (
    GraphicsImportComponent    ci,
    RgnHandle                  *clipRgn );
```

ci

The component instance that identifies your connection to the graphics importer component.

clipRgn

A handle to the `MacRegion` (IV–2303) structure that has been defined as the clipping region for the image. Returns `NIL` if there is no clipping region.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The caller must dispose of the returned region handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Setting Drawing Parameters” (V–2879)

Related Java Methods

```
quicktime.qd.Region.fromGraphicsImporter(),  
quicktime.std.image.GraphicsImporter.getClip()
```

See Also

For the corresponding set function, see GraphicsImportSetClip (I-650).

GraphicsImportGetColorSyncProfile

Returns a ColorSync profile for an imported image, if one is embedded in the image file.

```
ComponentResult GraphicsImportGetColorSyncProfile (  
    GraphicsImportComponent    ci,  
    Handle                     *profile );
```

ci

The component instance that identifies your connection to the graphics importer component.

profile

A pointer to receive a handle containing a ColorSync profile, or NIL if the image file does not contain one.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Discussion

Some graphics importers don’t implement this function. The caller is responsible for disposing of the returned handle.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Managing Graphis Importers” (V-2876)

GraphicsImportGetDataFile

Returns the file containing the graphics data for an imported image.

```
ComponentResult GraphicsImportGetDataFile (  
    GraphicsImportComponent    ci,  
    FSSpec                     *theFile );
```

GraphicsImportGetDataFile

ci

The component instance that identifies your connection to the graphics importer component.

theFile

A pointer in which to return the FSSpec (IV–2262) structure of the file containing the graphics data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. If the data source is not a file, the function returns `paramErr`.

Discussion

Use this function to get the file system specification record for the file where the imported graphics data resides.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides `GraphicsImportGetDataFile`, these functions include `GraphicsImportSetDataFile` (I–651), `GraphicsImportSetDataHandle` (I–652), `GraphicsImportGetDataHandle` (I–623), `GraphicsImportSetDataReference` (I–653), `GraphicsImportSetDataReferenceOffsetAndLimit` (I–654), and `GraphicsImportGetDataReferenceOffsetAndLimit` (I–627). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don’t need to call them if you use one of the `GetGraphicsImporter...` functions such as `GetGraphicsImporterForDataRef` (I–412). The `GetGraphicsImporter...` functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying a Graphics Import Data Source” (V–2882)

Related Java Methods

`quicktime.io.QTFile.fromGraphicsImporter()`,
`quicktime.std.image.GraphicsImporter.getDataFile()`

See Also

For the corresponding set function, see `GraphicsImportSetDataFile` (I–651).

GraphicsImportGetDataHandle

Returns a handle to imported graphics data.

```
ComponentResult GraphicsImportGetDataHandle (
    GraphicsImportComponent    ci,
    Handle                     *h );
```

ci

The component instance that identifies your connection to the graphics importer component.

h

A pointer in which to return a handle to the graphics data.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error. If the data source is not a handle, the function returns paramErr.

Discussion

You use this function to get the handle that the graphics data resides in. The handle belongs to the component instance. You shouldn’t dispose of it.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides GraphicsImportGetDataHandle, these functions include GraphicsImportSetDataFile (I–651), GraphicsImportSetDataHandle (I–652), GraphicsImportGetDataFile (I–621), GraphicsImportSetDataReference (I–653), GraphicsImportSetDataReferenceOffsetAndLimit (I–654), and GraphicsImportGetDataReferenceOffsetAndLimit (I–627). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don’t need to call them if you use one of the GetGraphicsImporter... functions such as GetGraphicsImporterForDataRef (I–412). The GetGraphicsImporter... functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Specifying a Graphics Import Data Source” (V–2882)

Related Java Methods

```
quicktime.std.image.GraphicsImporter.getDataHandle(),
quicktime.util.QTHandle.fromGraphicsImporterData()
```

GraphicsImportGetDataOffsetAndSize

See Also

For the corresponding set function, see `GraphicsImportSetDataHandle` (I-652).

GraphicsImportGetDataOffsetAndSize

Returns the offset and size of the compressed image data within an imported image file.

```
ComponentResult GraphicsImportGetDataOffsetAndSize (  
    GraphicsImportComponent    ci,  
    unsigned long               *offset,  
    unsigned long               *size );
```

ci

The component instance that identifies your connection to the graphics importer component.

offset

A pointer to a value describing the byte offset of the image data from the beginning of the data source.

size

A pointer to a value describing the size of the image data in bytes.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This function returns the offset and size of the actual image data within the data source. By default, the offset returned is 0 and the size returned is the size of the file. However, some graphics import components will override this function to skip over unneeded information at the beginning or end of the file.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting Image Characteristics” (V-2878), “Managing Graphis Importers” (V-2876)

Related Java Methods

```
quicktime.std.image.GraphicsImporter.getDataOffset(),  
quicktime.std.image.GraphicsImporter.getDataSize()
```

See Also

See `GraphicsImportGetDataOffsetAndSize64` (I-625). Format-specific graphics importers always implement `GraphicsImportGetImageDescription` (I-638) and may optionally implement the `GraphicsImportGetNaturalBounds` (I-642), `GraphicsImportValidate` (I-665), `GraphicsImportGetMetaData` (I-640), and `GraphicsImportDoesDrawAllPixels` (I-613), as well as `GraphicsImportGetDataOffsetAndSize`.

GraphicsImportGetDataOffsetAndSize64

Provides a 64-bit version of `GraphicsImportGetDataOffsetAndSize`.

```
ComponentResult GraphicsImportGetDataOffsetAndSize64 (
    GraphicsImportComponent ci,
    wide *offset,
    wide *size );
```

ci

The component instance that identifies your connection to the graphics importer component.

offset

A pointer to a value describing the byte offset of the image data.

size

A pointer to the size of the data, in bytes.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

Format-specific importers may delegate this function, in which case the base importer’s implementation will call the 32-bit equivalent, `GraphicsImportGetDataOffsetAndSize` (I-624). If neither function is implemented by the format-specific importer, then both functions will return an offset of 0 and the full size of the data reference, taking into account any data reference offset and limit.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V-2876)

See Also

See `GraphicsImportGetDataOffsetAndSize` (I-624).

GraphicsImportGetDataReference

Returns a data reference to imported graphics data.

```
ComponentResult GraphicsImportGetDataReference (  
    GraphicsImportComponent    ci,  
    Handle                     *dataRef,  
    OSType                     *dataReType );
```

ci

The component instance that identifies your connection to the graphics importer component.

dataRef

A pointer in which to return a QuickTime data reference. If you don't want this information, pass NIL.

dataReType

A pointer to receive the type of the data reference; see “Data References” (IV–2661). If you don't want this information, pass NIL.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

You use this function to get the data reference that the graphics data resides in. The GraphicsImportGetDataHandle (I–623) and GraphicsImportGetDataFile (I–621) functions call GraphicsImportGetDataReference and then manipulate the result accordingly. The caller should dispose of the returned dataRef.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides GraphicsImportGetDataReference, these functions include GraphicsImportSetDataFile (I–651), GraphicsImportSetDataHandle (I–652), GraphicsImportGetDataFile (I–621), GraphicsImportSetDataReference (I–653), GraphicsImportSetDataReferenceOffsetAndLimit (I–654), and GraphicsImportGetDataReferenceOffsetAndLimit (I–627). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don't need to call them if you use one of the GetGraphicsImporter... functions such as GetGraphicsImporterForDataRef (I–412). The GetGraphicsImporter... functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying a Graphics Import Data Source” (V–2882)

Related Java Methods

`quicktime.std.image.GraphicsImporter.getDataReferenceType()`

See Also

For the corresponding set function, see `GraphicsImportSetDataReference` (I–653).

GraphicsImportGetDataReferenceOffsetAndLimit

Returns the data reference starting offset and data size limit for an imported image.

```
ComponentResult GraphicsImportGetDataReferenceOffsetAndLimit (  
    GraphicsImportComponent    ci,  
    unsigned long              *offset,  
    unsigned long              *limit );
```

ci

The component instance that identifies your connection to the graphics importer component.

offset

A pointer to a value specifying the byte offset of the image data from the beginning of the data reference.

limit

The offset of the byte following the last byte of the image data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns the values set by the `GraphicsImportSetDataReferenceOffsetAndLimit` (I–654) function. By default, the offset is 0 and the limit is `MaxInt` ($2^{32} - 1$).

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides

`GraphicsImportGetDataReferenceOffsetAndLimit`, these functions include `GraphicsImportSetDataFile` (I–651), `GraphicsImportSetDataHandle` (I–652), `GraphicsImportGetDataFile` (I–621), `GraphicsImportSetDataReference` (I–653), `GraphicsImportSetDataReferenceOffsetAndLimit` (I–654), and

GraphicsImportGetDataReferenceOffsetAndLimit64

`GraphicsImportGetDataReference` (I–626). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don’t need to call them if you use one of the `GetGraphicsImporter...` functions such as `GetGraphicsImporterForDataRef` (I–412). The `GetGraphicsImporter...` functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V–2876), “Specifying a Graphics Import Data Source” (V–2882)

See Also

See `GraphicsImportGetDataReferenceOffsetAndLimit64` (I–628). For the corresponding set function, see `GraphicsImportSetDataReferenceOffsetAndLimit` (I–654).

GraphicsImportGetDataReferenceOffsetAndLimit64

Provides a 64-bit version of `GraphicsImportGetDataReferenceOffsetAndLimit` (I–627).

```
ComponentResult GraphicsImportGetDataReferenceOffsetAndLimit64 (
    GraphicsImportComponent    ci,
    wide                       *offset,
    wide                       *limit );
```

ci

The component instance that identifies your connection to the graphics importer component.

offset

A pointer to receive a value specifying the offset of the byte data following the last byte of the image data.

limit

A pointer to the data limit.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The only difference between this function and `GraphicsImportGetDataReferenceOffsetAndLimit` (I-627) is that the `offset` parameter and the `limit` parameter are 64-bit integers instead of 32-bit integers.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V-2876)

See Also

See `GraphicsImportGetDataReferenceOffsetAndLimit` (I-627). For the corresponding set function, see `GraphicsImportSetDataReferenceOffsetAndLimit64` (I-656).

GraphicsImportGetDefaultClip

Returns the default clipping region for an imported image, if one is stored there.

```
ComponentResult GraphicsImportGetDefaultClip (
    GraphicsImportComponent    ci,
    RgnHandle                   *defaultRgn );
```

ci

The component instance that identifies your connection to the graphics importer component.

defaultRgn

A pointer to a handle to a `MacRegion` (IV-2303) structure to receive the default clipping region.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error. Returns `badComponentSelector` if there is no clipping region.

Special Considerations

Most graphics importers don’t implement this function. The caller is responsible for disposing of the returned region.

Version Notes

Introduced in QuickTime 4.

GraphicsImportGetDefaultGraphicsMode

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V–2876)

GraphicsImportGetDefaultGraphicsMode

Returns the default graphics mode for an imported image, if one is stored there.

```
ComponentResult GraphicsImportGetDefaultGraphicsMode (  
    GraphicsImportComponent    ci,  
    long                       *defaultGraphicsMode,  
    RGBColor                   *defaultOpColor );
```

ci

The component instance that identifies your connection to the graphics importer component.

defaultGraphicsMode

A pointer to receive the graphics mode; see “Graphics Transfer Modes” (IV–2670).

defaultOpColor

A pointer to receive a color; see “Color Constants” (IV–2657).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. If this function returns `badComponentSelector`, you should assume a mode of **ditherCopy**.

Special Considerations

Most graphics importers don’t implement this function.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V–2876)

GraphicsImportGetDefaultMatrix

Returns the default matrix for an imported image, if one is stored there.

```
ComponentResult GraphicsImportGetDefaultMatrix (  
    GraphicsImportComponent    ci,
```

```
MatrixRecord          *defaultMatrix );
```

ci

The component instance that identifies your connection to the graphics importer component.

defaultMatrix

Receives a matrix record.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If this function returns `badComponentSelector`, you should assume an identity matrix.

Special Considerations

Most graphics importers don’t implement this function.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V–2876)

GraphicsImportGetDefaultSourceRect

Returns the default source rectangle for an imported image, if one is stored there.

```
ComponentResult GraphicsImportGetDefaultSourceRect (  
    GraphicsImportComponent    ci,  
    Rect                      *defaultSourceRect );
```

ci

The component instance that identifies your connection to the graphics importer component.

defaultSourceRect

Pointer to receive a `Rect` (IV–2399) structure that describes the default source rectangle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. If this function returns `badComponentSelector`, the source rectangle is equal to the image’s natural bounds.

GraphicsImportGetDestRect

Special Considerations

Most graphics importers don't implement this function.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: "Managing Graphis Importers" (V-2876)

GraphicsImportGetDestRect

Returns the destination rectangle for an imported image.

```
ComponentResult GraphicsImportGetDestRect (
    GraphicsImportComponent    ci,
    Rect                       *destRect );
```

ci

The component instance that identifies your connection to the graphics importer component.

destRect

A pointer to receive a `Rect` (IV-2399) structure that describes the destination rectangle.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

If the source rectangle is equal to the natural bounds, this function is equivalent to `GraphicsImportGetBoundsRect` (I-619).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: "Managing Graphis Importers" (V-2876)

See Also

For the corresponding set function, see `GraphicsImportSetDestRect` (I-657).

GraphicsImportGetExportImageTypeList

Returns information about available export formats for a graphics importer.

GraphicsImportGetExportSettingsAsAtomContainer

```
ComponentResult GraphicsImportGetExportImageTypeList (
    GraphicsImportComponent ci,
    void *qtAtomContainerPtr );
```

ci

The component instance that identifies your connection to the graphics importer component.

qtAtomContainerPtr

A pointer to a QuickTime atom container that is to receive the type list.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function creates and returns a QuickTime atom container of type 'expo' (IV–2535) containing information about the file types that can be exported by the graphics import component. For each file type, the atom container contains the following child atoms: 'ftyp' (IV–2539), the exported file type; 'mime' (IV–2561), the MIME type for this format (optional); 'ext ' (IV–2536), the suggested file extension for this format; and 'desc' (IV–2528), a human-readable name for this format. The 'ftyp' atom contains an OSType; the other atoms contain nonterminated strings.

Special Considerations

It is the responsibility of the caller to dispose of the 'expo' atom container.

Version Notes

In QuickTime 3, the supported export file types are `kQTFileTypePicture`, `kQTFileTypeQuickTimeImage`, `kQTFileTypeBMP`, `kQTFileTypeJPEG`, and `kQTFileTypePhotoShop`. In QuickTime 4, the generic graphics importer builds this atom container from the values returned by the installed graphics exporter components.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Saving Image Files” (V–2881)

Related Java Methods

```
quicktime.std.image.GraphicsImporter.getExportImageTypeList(),
quicktime.std.movies.AtomContainer.fromGraphicsImporterExportImage()
```

GraphicsImportGetExportSettingsAsAtomContainer

Retrieves settings for image files exported by the graphics importer.

GraphicsImportGetFlags

```
ComponentResult GraphicsImportGetExportSettingsAsAtomContainer (
    GraphicsImportComponent    ci,
    void                        *qtAtomContainerPtr );
```

ci

The component instance that identifies your connection to the graphics importer component.

qtAtomContainerPtr

A pointer to a QuickTime atom container that is to receive the settings information.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function creates and returns a new QuickTime atom container which holds information about how images will be saved by GraphicsImportExportImageFile (I–616).

Special Considerations

It is the responsibility of the caller to dispose of this atom container.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Saving Image Files” (V–2881)

Related Java Methods

quicktime.std.image.GraphicsImporter.getExportSettingsAsAtomContainer(),

quicktime.std.movies.AtomContainer.fromGraphicsImporterExportSettings()

GraphicsImportGetFlags

Returns the current flags of a graphics importer component.

```
ComponentResult GraphicsImportGetFlags (
    GraphicsImportComponent    ci,
    long                        *flags );
```

ci

The component instance that identifies your connection to a graphics importer component.

flags

Pointer to a long integer to receive the current flags (see below).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

flags Constants

kGraphicsImporterDontDoGammaCorrection

The graphics importer does not perform gamma correction.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Managing Graphis Importers” (V–2876)

See Also

For the corresponding set function, see GraphicsImportSetFlags (I–658).

GraphicsImportGetGraphicsMode

Returns the graphics transfer mode for an imported image.

```
ComponentResult GraphicsImportGetGraphicsMode (
    GraphicsImportComponent  ci,
    long                    *graphicsMode,
    RGBColor                 *opColor );
```

ci

The component instance that identifies your connection to the graphics importer component.

graphicsMode

A pointer to a long integer; see “Graphics Transfer Modes” (IV–2670). The function returns the QuickDraw graphics transfer mode setting for the image. Set to NIL if you are not interested in this information.

opColor

A pointer to an RGBColor (IV–2403) structure. The function returns the color currently specified for blend and transparent operations. Set to NIL if you are not interested in this information.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

GraphicsImportGetGWorld

Discussion

Use this function to find out the current graphics transfer mode and color to use for blending and transparent operations. The default graphics mode is **ditherCopy** and the default `opColor` is 50% gray.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Setting Drawing Parameters” (V–2879)

Related Java Methods

`quicktime.std.image.GraphicsImporter.getGraphicsMode()`

See Also

For the corresponding set function, see `GraphicsImportSetGraphicsMode` (I–659).

GraphicsImportGetGWorld

Returns the current graphics port and device for drawing an imported image.

```
ComponentResult GraphicsImportGetGWorld (
    GraphicsImportComponent    ci,
    CGrafPtr                   *port,
    GDHandle                    *gd );
```

ci

The component instance that identifies your connection to the graphics importer component.

port

Returns a pointer to the `CGrafPort` (IV–2168) structure for the current destination graphics port. Set to `NIL` if you are not interested in this information.

gd

Returns a pointer to the `GDevice` (IV–2264) structure for the destination graphics device. Set to `NIL` if you are not interested in this information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns the graphics port and device that will be used to draw the image. The graphics world is initialized to the current port and device when the graphics importer component is opened.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Drawing Imported Images” (V–2880)

Related Java Methods

`quicktime.std.image.GraphicsImporter.getGWorld()`,

`quicktime.qd.QDGraphics.fromGraphicsImporter()`

See Also

For the corresponding set function, see `GraphicsImportSetGWorld` (I–660).

GraphicsImportGetImageCount

Returns the number of images in an imported image file.

```
ComponentResult GraphicsImportGetImageCount (
    GraphicsImportComponent  ci,
    unsigned long            *imageCount );
```

ci

The component instance that identifies your connection to the graphics importer component.

imageCount

Points to a variable to receive the number of images.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Most image file formats don’t support multiple images. Of the image formats supported by QuickTime 4, however, TIFF files can support multiple images, Photoshop files can contain multiple layers and FlashPix files can contain multiple resolutions. The **base graphics importer** returns a count of 1.

Special Considerations

Format-specific importers for multiple-image formats should override this function; other importers should delegate it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V–2876)

GraphicsImportGetImageDescription

GraphicsImportGetImageDescription

Returns image description information for an imported image.

```
ComponentResult GraphicsImportGetImageDescription (  
    GraphicsImportComponent    ci,  
    ImageDescriptionHandle     *desc );
```

ci

The component instance that identifies your connection to the graphics importer component.

desc

Points to a handle to an `ImageDescription` (IV–2285) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns an `ImageDescription` (IV–2285) structure containing information such as the format of the compressed data, its bit depth, natural bounds, and resolution.

Special Considerations

The caller is responsible for disposing of the returned image description handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting Image Characteristics” (V–2878)

Related Java Methods

`quicktime.std.image.GraphicsImporter.getImageDescription()`,

`quicktime.std.image.ImageDescription.fromGraphicsImporter()`

See Also

Format-specific graphics importers always implement `GraphicsImportGetImageDescription` and may optionally implement `GraphicsImportGetNaturalBounds` (I–642), `GraphicsImportGetDataOffsetAndSize` (I–624), `GraphicsImportValidate` (I–665), `GraphicsImportGetMetaData` (I–640), and `GraphicsImportDoesDrawAllPixels` (I–613).

GraphicsImportGetImageIndex

Returns the current image index for an imported image.

```
ComponentResult GraphicsImportGetImageIndex (
    GraphicsImportComponent    ci,
    unsigned long              *imageIndex );
```

ci

The component instance that identifies your connection to the graphics importer component.

imageIndex

Points to a variable to receive the image index. Image indexes are one-based; 0 is considered a special index by some importers, and treated the same as 1 by others. The default image index is 1.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The **base graphics importer** implements this function. Format-specific importers should delegate it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V–2876)

See Also

For the corresponding set function, see `GraphicsImportSetImageIndex` (I–661).

GraphicsImportGetMatrix

Returns the transformation matrix to be used for drawing an imported image.

```
ComponentResult GraphicsImportGetMatrix (
    GraphicsImportComponent    ci,
    MatrixRecord               *matrix );
```

ci

The component instance that identifies your connection to the graphics importer component.

matrix

A pointer to a `MatrixRecord` (IV–2304) structure that defines the transformation matrix that applies to the image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

GraphicsImportGetMetaData

Discussion

The transformation matrix is initialized to the identity matrix when the graphics import component is instantiated.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Setting Drawing Parameters” (V–2879)

Related Java Methods

`quicktime.std.image.GraphicsImporter.getMatrix()`

See Also

For the corresponding set function, see `GraphicsImportSetMatrix` (I–661).

GraphicsImportGetMetaData

Extracts user data from an imported image file.

```
ComponentResult GraphicsImportGetMetaData (
    GraphicsImportComponent    ci,
    void                      *userData );
```

ci

The component instance that identifies your connection to the graphics importer component.

userData

A pointer to a `UserDataRecord` (IV–2496) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You may create a new user data structure by calling `NewUserData` (II–1124). Alternatively, you can obtain a pointer to an existing one by calling `GetMovieUserData` (I–499), `GetTrackUserData` (I–546) or `GetMediaUserData` (I–456). If the user data passed to `GraphicsImportGetMetaData` belongs to a movie, track or media, then whatever user data is extracted will be added to that movie, track or media.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting Image Characteristics” (V–2878)

Related Java Methods

`quicktime.std.image.GraphicsImporter.getMetaData()`

See Also

Format-specific graphics importers always implement `GraphicsImportGetImageDescription` (I–638) and may optionally implement `GraphicsImportGetNaturalBounds` (I–642), `GraphicsImportGetDataOffsetAndSize` (I–624), `GraphicsImportValidate` (I–665), and `GraphicsImportDoesDrawAllPixels` (I–613), as well as `GraphicsImportGetMetaData`.

GraphicsImportGetMIMETypeList

Returns a list of MIME types supported by the graphics importer component.

```
ComponentResult GraphicsImportGetMIMETypeList (
    GraphicsImportComponent    ci,
    void                       *qtAtomContainerPtr );
```

ci

Specifies an instance of a graphics importer component.

qtAtomContainerPtr

A pointer to a QuickTime atom container of type 'mime' (IV–2561) that contains a list of MIME types supported by the graphics import component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your graphics import component can support MIME types that correspond to graphics formats it supports. To make a list of these MIME types available to applications or other software, it must implement `GraphicsImportGetMIMETypeList`. To indicate that your graphics import component supports this function, set the `hasMovieImportMIMEList` flag in the `componentFlags` field of your component’s `ComponentDescription` (IV–2212) structure.

Special Considerations

This function does not access any file-specific information.

Version Notes

Introduced in QuickTime 3 or earlier.

GraphicsImportGetNaturalBounds

Programming Info

C interface file: `ImageCompression.h`
Programming summary: “Getting MIME Types” (V–2881)

Related Java Methods

`quicktime.std.image.GraphicsImporter.getMIMETypeList()`,
`quicktime.std.movies.AtomContainer.fromGraphicsImporterMIME()`

GraphicsImportGetNaturalBounds

Returns the bounding rectangle of an imported image.

```
ComponentResult GraphicsImportGetNaturalBounds (  
    GraphicsImportComponent ci,  
    Rect *naturalBounds );
```

ci

The component instance that identifies your connection to the graphics importer component.

naturalBounds

A pointer to a `Rect` (IV–2399) structure that describes the size of the bounding rectangle for the image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Use this function to determine the native size of the image associated with a graphics importer component. The natural bounds are always zero-based. This is a convenience function that simply calls `GraphicsImportGetImageDescription` (I–638) and extracts the width and height fields.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`
Programming summary: “Getting Image Characteristics” (V–2878)

Related Java Methods

`quicktime.std.image.GraphicsImporter.getNaturalBounds()`

See Also

Format-specific graphics importers always implement `GraphicsImportGetImageDescription` (I–638) and may optionally implement `GraphicsImportGetDataOffsetAndSize` (I–624), `GraphicsImportValidate` (I–665),

GraphicsImportGetMetaData (I–640), and GraphicsImportDoesDrawAllPixels (I–613), as well as GraphicsImportGetNaturalBounds.

GraphicsImportGetProgressProc

Returns the current **progress function** for a graphics import operation.

```
ComponentResult GraphicsImportGetProgressProc (
    GraphicsImportComponent    ci,
    ICMProgressProcRecordPtr    progressProc );
```

ci

The component instance that identifies your connection to the graphics importer component.

progressProc

A pointer to an ICMProgressProc (III–2093) callback.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

By default, graphics import components have no **progress functions**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Setting Drawing Parameters” (V–2879)

See Also

For the corresponding set function, see GraphicsImportSetProgressProc (I–662).

GraphicsImportGetQuality

Returns the image quality value for an imported image.

```
ComponentResult GraphicsImportGetQuality (
    GraphicsImportComponent    ci,
    CodecQ                     *quality );
```

ci

The component instance that identifies your connection to the graphics importer component.

GraphicsImportGetQuality

quality

A pointer to a constant (see below) that defines the currently specified quality value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

quality Constants

`codecMinQuality`

The minimum valid value for a CodecQ field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a CodecQ field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

The quality value indicates how precisely the decompressor will decompress the image data. Some decompressors may choose to ignore some image data to improve decompression speed.

Version Notes

With QuickTime 3 the default quality is `codecHighQuality`.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Setting Drawing Parameters” (V–2879)

Related Java Methods

`quicktime.std.image.GraphicsImporter.getQuality()`

See Also

For the corresponding set function, see `GraphicsImportSetQuality` (I–663).

GraphicsImportGetSourceRect

Returns the current source rectangle for an imported image.

```
ComponentResult GraphicsImportGetSourceRect (
    GraphicsImportComponent    ci,
    Rect                      *sourceRect );
```

ci

The component instance that identifies your connection to the graphics importer component.

sourceRect

A pointer to a Rect (IV-2399) structure that defines the source rectangle currently specified for the image.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Discussion

This function returns the current source rectangle, as specified by GraphicsImportSetSourceRect (I-664). The default source rectangle is the image’s natural bounds.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Setting Drawing Parameters” (V-2879)

Related Java Methods

`quicktime.std.image.GraphicsImporter.getSourceRect()`

See Also

For the corresponding set function, see GraphicsImportSetSourceRect (I-664).

GraphicsImportReadData

Reads imported image data.

```
ComponentResult GraphicsImportReadData (
    GraphicsImportComponent    ci,
    void                      *dataPtr,
    unsigned long              dataOffset,
    unsigned long              dataSize );
```

GraphicsImportReadData64

ci

The component instance that identifies your connection to the graphics importer component.

dataPtr

A pointer to a memory block to receive the data.

dataOffset

The offset of the image data within the data reference. The function begins reading image data from this offset.

dataSize

The number of bytes of image data to read.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function communicates with the appropriate data handler to retrieve image data. Typically, only developers of graphics importer components will need to use this function. This function should always be used to retrieve data from the data source, rather than reading the data directly. This function automatically honors any offset and limit values set with `GraphicsImportSetDataReferenceOffsetAndLimit` (I–654). For instance, if the offset is set to 100 and `GraphicsImportReadData` is called to read bytes from `dataOffset` 5, it will return bytes starting at actual offset 105.

Special Considerations

This function is used by format-specific graphics import components to read data from the data source. It is implemented by the **base graphics importer**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V–2876), “Retrieving Graphics Import Image Data” (V–2883)

Related Java Methods

`quicktime.std.image.GraphicsImporter.readData()`

See Also

See `GraphicsImportReadData64` (I–646).

GraphicsImportReadData64

Provides a 64-bit version of `GraphicsImportReadData` (I–645).

```
ComponentResult GraphicsImportReadData64 (  
    GraphicsImportComponent    ci,  
    void                       *dataPtr,  
    const wide                 *dataOffset,  
    unsigned long              dataSize );
```

ci

The component instance that identifies your connection to the graphics importer component.

dataPtr

A pointer to a memory block to receive the data.

dataOffset

A pointer to the offset of the image data within the data reference.

dataSize

The number of bytes of image data to read.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Discussion

This function is a 64-bit analog of GraphicsImportReadData (I-645). Format-specific importers may call either version.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Managing Graphis Importers” (V-2876)

See Also

See GraphicsImportReadData (I-645).

GraphicsImportSaveAsPicture

Creates a QuickDraw **picture file** for an imported image.

```
ComponentResult GraphicsImportSaveAsPicture (  
    GraphicsImportComponent    ci,  
    const FSSpec               *fss,  
    ScriptCode                 scriptTag );
```

ci

The component instance that identifies your connection to the graphics importer component.

GraphicsImportSaveAsQuickTimeImageFile

fss

A pointer to an FSSpec (IV–2262) structure that defines the file to receive the image.

scriptTag

The script system in which the file name is to be displayed; see “Localization Codes” (IV–2673). If you have established the name and location of the file using one of the **Standard File Package** functions, use the script code returned in the reply record (reply.sfScript). Otherwise, specify the system script by setting the scriptTag parameter to the value smSystemScript. See *Inside Macintosh: Files* (listed in the **bibliography**) for more information about script specifications.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function creates a new QuickDraw **picture file** containing the image currently in use by the graphics import component. If possible, the image will remain in the compressed format. For example, if the image is from a JFIF file, the picture will contain compressed JPEG data. Applications that only wish to draw the image can use GraphicsImportDraw (I–616).

Special Considerations

Graphics import components can save data in several formats, including QuickDraw pictures and QuickTime Image files. This capability is only needed by applications that perform file format translation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Saving Image Files” (V–2881)

Related Java Methods

quicktime.std.image.GraphicsImporter.saveAsPicture()

GraphicsImportSaveAsQuickTimeImageFile

Creates a QuickTime Image file of an imported image.

```
ComponentResult GraphicsImportSaveAsQuickTimeImageFile (
    GraphicsImportComponent    ci,
    const FSSpec                *fss,
    ScriptCode                  scriptTag );
```

`ci`

The component instance that identifies your connection to the graphics importer component.

`fss`

A pointer to the `FSSpec` (IV–2262) that defines the file to receive the image.

`scriptTag`

The script system in which the file name is to be displayed; see “Localization Codes” (IV–2673). If you have established the name and location of the file using one of the **Standard File Package** functions, use the script code returned in the reply record (`reply.sfScript`). Otherwise, specify the system script by setting the `scriptTag` parameter to the value `smSystemScript`. See *Inside Macintosh: Files* (listed in the **bibliography**) for more information about script specifications.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function creates a new QuickTime Image file containing the image currently in use by the graphics import component. If possible, the image remains in the compressed format. For example, if the image is from a JFIF file, the QuickTime Image file will contain compressed JPEG data.

Special Considerations

Graphics import components can save data in several formats, including QuickDraw pictures and QuickTime Image files. This capability is only needed by applications that perform file format translation. Applications that only wish to draw the image can use the `GraphicsImportDraw` (I–616) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Saving Image Files” (V–2881)

Related Java Methods

`quicktime.std.image.GraphicsImporter.saveAsQuickTimeImageFile()`

GraphicsImportSetBoundsRect

Defines the rectangle in which to draw an imported image.

```
ComponentResult GraphicsImportSetBoundsRect (
    GraphicsImportComponent  ci,
```

GraphicsImportSetClip

```
const Rect          *bounds );
```

ci

The component instance that identifies your connection to the graphics importer component.

bounds

A pointer to a Rect (IV–2399) structure that describes the bounding rectangle into which the image will be drawn.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

You use this function to define the rectangle into which the graphics image should be drawn. The function creates a transformation matrix to map the image’s natural bounds to the specified bounds and then calls GraphicsImportSetMatrix (I–661).

Special Considerations

Because this function affects the transformation matrix, you should use GraphicsImportSetMatrix (I–661) instead of this function if you also need to specify more complex transformations of the matrix.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Setting Drawing Parameters” (V–2879)

Related Java Methods

quicktime.std.image.GraphicsImporter.setBoundsRect()

See Also

For the corresponding get function, see GraphicsImportGetBoundsRect (I–619). If the source rectangle is equal to the natural bounds, GraphicsImportSetBoundsRect is equivalent to GraphicsImportSetDestRect (I–657).

GraphicsImportSetClip

Defines the clipping region for drawing an imported image.

```
ComponentResult GraphicsImportSetClip (  
    GraphicsImportComponent    ci,  
    RgnHandle                  clipRgn );
```


ci

The component instance that identifies your connection to the graphics importer component.

clipRgn

A handle to a `MacRegion` (IV–2303) structure that defines the clipping region in the destination coordinate system. Set to `NIL` to disable clipping. The graphics import component makes a copy of this region.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Because all drawing operations ignore the port clipping region, you must use this function to clip an image. The graphics importer component draws only that portion of the image that lies within the specified clipping region.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Setting Drawing Parameters” (V–2879)

Related Java Methods

`quicktime.std.image.GraphicsImporter.setClip()`

See Also

For the corresponding get function, see `GraphicsImportGetClip` (I–620).

GraphicsImportSetDataFile

Specifies the file that contains imported graphics data.

```
ComponentResult GraphicsImportSetDataFile (
    GraphicsImportComponent    ci,
    const FSSpec                *theFile );
```

ci

The component instance that identifies your connection to the graphics importer component.

theFile

A pointer to an `FSSpec` (IV–2262) structure that defines the file containing the graphics data. The returned file will be opened for read access.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

GraphicsImportSetDataHandle

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides `GraphicsImportSetDataFile`, these functions include `GraphicsImportGetDataFile` (I–621), `GraphicsImportSetDataHandle` (I–652), `GraphicsImportGetDataHandle` (I–623), `GraphicsImportSetDataReference` (I–653), `GraphicsImportSetDataReferenceOffsetAndLimit` (I–654), and `GraphicsImportGetDataReferenceOffsetAndLimit` (I–627). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don't need to call them if you use one of the `GetGraphicsImporter...` functions such as `GetGraphicsImporterForDataRef` (I–412). The `GetGraphicsImporter...` functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Specifying a Graphics Import Data Source” (V–2882)

Related Java Methods

`quicktime.std.image.GraphicsImporter.setDataFile()`

See Also

For the corresponding get function, see `GraphicsImportGetDataFile` (I–621).

GraphicsImportSetDataHandle

Specifies the handle that references imported graphics data.

```
ComponentResult GraphicsImportSetDataHandle (
    GraphicsImportComponent  ci,
    Handle                   h );
```

ci

The component instance that identifies your connection to the graphics importer component.

h

Specifies a handle containing graphics data. The format of the data in the handle is the same as that found in a file.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The graphics importer component doesn't make a copy of this data. Therefore, you must not dispose this handle until the graphics importer has been closed.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides `GraphicsImportSetDataHandle`, these functions include `GraphicsImportGetDataFile` (I-621), `GraphicsImportSetDataFile` (I-651), `GraphicsImportGetDataHandle` (I-623), `GraphicsImportSetDataReference` (I-653), `GraphicsImportSetDataReferenceOffsetAndLimit` (I-654), and `GraphicsImportGetDataReferenceOffsetAndLimit` (I-627). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don't need to call them if you use one of the `GetGraphicsImporter...` functions such as `GetGraphicsImporterForDataRef` (I-412). The `GetGraphicsImporter...` functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: "Specifying a Graphics Import Data Source" (V-2882)

Related Java Methods

`quicktime.std.image.GraphicsImporter.setDataHandle()`

See Also

For the corresponding get function, see `GraphicsImportGetDataHandle` (I-623).

GraphicsImportSetDataReference

Specifies the data reference for imported graphics data.

```
ComponentResult GraphicsImportSetDataReference (
    GraphicsImportComponent  ci,
    Handle                   dataRef,
    OSType                   dataReType );
```

`ci`

The component instance that identifies your connection to the graphics importer component.

GraphicsImportSetDataReferenceOffsetAndLimit

dataRef

A handle to a QuickTime data reference.

dataReType

The data reference type. See “Data References” (IV–2661).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Applications typically don’t use this function. The GraphicsImportSetDataFile (I–651) and GraphicsImportSetDataHandle (I–652) functions both call this function, with the appropriate data reference and data reference type. This function makes a copy of the passed data reference, so it is safe to dispose of the handle immediately after the call.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides GraphicsImportSetDataReference, these functions include GraphicsImportGetDataFile (I–621), GraphicsImportSetDataHandle (I–652), GraphicsImportGetDataHandle (I–623), GraphicsImportSetDataFile (I–651), GraphicsImportSetDataReferenceOffsetAndLimit (I–654), and GraphicsImportGetDataReferenceOffsetAndLimit (I–627). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don’t need to call them if you use one of the GetGraphicsImporter... functions such as GetGraphicsImporterForDataRef (I–412). The GetGraphicsImporter... functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Specifying a Graphics Import Data Source” (V–2882)

Related Java Methods

`quicktime.std.image.GraphicsImporter.setDataReference()`

See Also

For the corresponding get function, see GraphicsImportGetDataReference (I–626).

GraphicsImportSetDataReferenceOffsetAndLimit

Specifies the data reference starting offset and data size limit for an imported image.

```
ComponentResult GraphicsImportSetDataReferenceOffsetAndLimit (  
    GraphicsImportComponent    ci,  
    unsigned long              offset,  
    unsigned long              limit );
```

ci

The component instance that identifies your connection to the graphics importer component.

offset

The byte offset of the image data from the beginning of the data reference.

limit

A pointer to a value specifying the offset of the byte following the last byte of the image data.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

A data reference typically refers to an entire file. However, there are times when the data being referenced is embedded in a larger file. In these cases, it is necessary to indicate where the data begins in the data reference and where it ends. This function lets you specify the starting offset and ending offset. All requests to read data are then relative to the specified offset, and are pinned to the data size, so the graphics import component cannot accidentally read outside the end (or beginning) of the segment.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides

GraphicsImportSetDataReferenceOffsetAndLimit, these functions include GraphicsImportGetDataFile (I–621), GraphicsImportSetDataHandle (I–652), GraphicsImportGetDataHandle (I–623), GraphicsImportSetDataReference (I–653), GraphicsImportSetDataFile (I–651), and

GraphicsImportGetDataReferenceOffsetAndLimit (I–627). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don’t need to call them if you use one of the GetGraphicsImporter... functions such as GetGraphicsImporterForDataRef (I–412). The GetGraphicsImporter... functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

GraphicsImportSetDataReferenceOffsetAndLimit64

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V–2876), “Specifying a Graphics Import Data Source” (V–2882)

See Also

See `GraphicsImportSetDataReferenceOffsetAndLimit64` (I–656). For the corresponding get function, see `GraphicsImportGetDataReferenceOffsetAndLimit` (I–627).

GraphicsImportSetDataReferenceOffsetAndLimit64

Provides a 64-bit version of `GraphicsImportSetDataReferenceOffsetAndLimit` (I–654).

```
ComponentResult GraphicsImportSetDataReferenceOffsetAndLimit64 (  
    GraphicsImportComponent    ci,  
    const wide                 *offset,  
    const wide                 *limit );
```

ci

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

offset

A pointer to a value specifying the byte offset of the image data from the beginning of the data source.

limit

A pointer to a value specifying the offset of the byte following the last byte of the image data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is a 64-bit analog of `GraphicsImportSetDataReferenceOffsetAndLimit` (I–654).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V–2876)

See Also

See GraphicsImportSetDataReferenceOffsetAndLimit (I–654). For the corresponding get function, see GraphicsImportGetDataReferenceOffsetAndLimit64 (I–628).

GraphicsImportSetDestRect

Sets the destination rectangle for a graphics import operation.

```
ComponentResult GraphicsImportSetDestRect (
    GraphicsImportComponent    ci,
    const Rect                 *destRect );
```

ci

The component instance that identifies your connection to the graphics importer component.

destRect

Points to a Rect (IV–2399) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Use this function to define the rectangle into which the extracted source rectangle should be drawn. This function creates a transformation matrix to map the source rectangle to the specified destination rectangle and then calls the GraphicsImportSetMatrix (I–661) function.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Managing Graphis Importers” (V–2876)

See Also

For the corresponding get function, see GraphicsImportGetDestRect (I–632). If the source rectangle is equal to the natural bounds, this function is equivalent to GraphicsImportSetBoundsRect (I–649).

GraphicsImportSetExportSettingsFromAtomContainer

Determines settings for the export of imported image files.

```
ComponentResult GraphicsImportSetExportSettingsFromAtomContainer (
```

GraphicsImportSetFlags

```
GraphicsImportComponent    ci,  
void                       *qtAtomContainer );
```

ci

The component instance that identifies your connection to the graphics importer component.

qtAtomContainer

A pointer to a QuickTime atom container that holds new settings information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function extracts export settings from a QuickTime atom container. These settings configure how images will be saved by `GraphicsImportExportImageFile` (I–616).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Saving Image Files” (V–2881)

Related Java Methods

`quicktime.std.image.GraphicsImporter.setExportSettingsFromAtomContainer()`

GraphicsImportSetFlags

Sets the flags for a graphics importer component.

```
ComponentResult GraphicsImportSetFlags (  
    GraphicsImportComponent    ci,  
    long                       flags );
```

ci

The component instance that identifies your connection to the graphics importer component.

flags

The new flags (see below) to use.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`kGraphicsImporterDontDoGammaCorrection`

The graphics importer will not perform gamma correction.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V–2876)

See Also

For the corresponding get function, see `GraphicsImportGetFlags` (I–634).

GraphicsImportSetGraphicsMode

Sets the graphics transfer mode for an imported image.

```
ComponentResult GraphicsImportSetGraphicsMode (
    GraphicsImportComponent  ci,
    long                     graphicsMode,
    const RGBColor           *opColor );
```

ci

The component instance that identifies your connection to the graphics importer component.

graphicsMode

The graphics transfer mode to use for drawing the image; see “Graphics Transfer Modes” (IV–2670).

opColor

A pointer to an `RGBColor` (IV–2403) structure that describes the color to use for blending and transparent operations.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Use this function to specify the graphics transfer mode and color to use for blending and transparent operations.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Setting Drawing Parameters” (V–2879)

GraphicsImportSetGWorld

Related Java Methods

`quicktime.std.image.GraphicsImporter.setGraphicsMode()`

See Also

For the corresponding get function, see `GraphicsImportGetGraphicsMode` (I–635).

GraphicsImportSetGWorld

Sets the graphics port and device for drawing an imported image.

```
ComponentResult GraphicsImportSetGWorld (
    GraphicsImportComponent    ci,
    CGrafPtr                  port,
    GDHandle                   gd );
```

ci

The component instance that identifies your connection to the graphics importer component.

port

A pointer to the `CGrafPort` (IV–2168) structure that defines the destination graphics port or graphics world. Set to `NIL` to use the current port.

gd

A handled to the `GDevice` (IV–2264) structure that defines the destination graphics device. Set to `NIL` to use the current device. If the *port* parameter specifies a graphics world, set this parameter to `NIL` to use that graphics world’s device.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The graphics world is initialized to the current port and device when the graphics importer component is opened. Use this function to select another port or device.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Drawing Imported Images” (V–2880)

Related Java Methods

`quicktime.std.image.GraphicsImporter.setGWorld()`

See Also

For the corresponding get function, see `GraphicsImportGetGWorld` (I–636).

GraphicsImportSetImageIndex

Specifies the image index for an imported image.

```
ComponentResult GraphicsImportSetImageIndex (
    GraphicsImportComponent    ci,
    unsigned long              imageIndex );
```

ci

The component instance that identifies your connection to the graphics importer component.

imageIndex

The image index. Image indexes are one-based; 0 is considered a special index by some importers, and treated the same as 1 by others. The default image index is 1.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

The **base graphics importer** ensures that the image index is no greater than the image count returned by `GraphicsImportGetImageCount` (I-637).

Special Considerations

The **base graphics importer** implements this function. Format-specific importers should delegate it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Graphis Importers” (V-2876)

See Also

For the corresponding get function, see `GraphicsImportGetImageIndex` (I-638).

GraphicsImportSetMatrix

Defines the transformation matrix to use for drawing an imported image.

```
ComponentResult GraphicsImportSetMatrix (
    GraphicsImportComponent    ci,
    const MatrixRecord         *matrix );
```

GraphicsImportSetProgressProc

ci

The component instance that identifies your connection to the graphics importer component.

matrix

A pointer to a matrix structure that specifies how to transform the image during decompression. For example, you can use a transformation matrix to scale or rotate the image. To set the matrix to identity, pass `NIL` in this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function establishes the transformation matrix to be applied to an image, which determines where and how it will be drawn.

Special Considerations

This function affects the bounding rectangle defined for the image. You can specify where an image will be drawn by setting either a transformation matrix or a bounding rectangle, but it is usually more convenient for applications to set a bounding rectangle using the `GraphicsImportSetBoundsRect` (I–649) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Setting Drawing Parameters” (V–2879)

Related Java Methods

`quicktime.std.image.GraphicsImporter.setMatrix()`

See Also

For the corresponding get function, see `GraphicsImportGetMatrix` (I–639).

GraphicsImportSetProgressProc

Installs a progress procedure to call while drawing an imported image.

```
ComponentResult GraphicsImportSetProgressProc (  
    GraphicsImportComponent    ci,  
    ICMProgressProcRecordPtr    progressProc );
```

ci

The component instance that identifies your connection to the graphics importer component.

`progressProc`

Points to an `ICMProgressProc` (III–2093) callback. If you pass a value of `-1`, QuickTime provides a standard **progress function**. If you want to remove the existing progress function, pass `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function sets a **progress function** that will be installed in the image decompression sequence used to draw the image.

Special Considerations

If your **progress function** does any drawing, you should take care to set a safe graphics state before doing so, and to restore the graphics state afterwards. In particular, the current graphics device may be an offscreen device.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Setting Drawing Parameters” (V–2879)

See Also

For the corresponding get function, see `GraphicsImportGetProgressProc` (I–643).

GraphicsImportSetQuality

Sets the image quality value for an imported image.

```
ComponentResult GraphicsImportSetQuality (
    GraphicsImportComponent  ci,
    CodecQ                   quality );
```

`ci`

The component instance that identifies your connection to the graphics importer component.

`quality`

Contains a constant (see below) that defines the desired image quality for decompression. Values for this parameter are on the same scale as compression quality.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

GraphicsImportSetSourceRect

quality Constants

`codecMinQuality`

The minimum valid value for a CodecQ field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a CodecQ field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

The quality parameter controls how precisely the decompressor decompresses the image data. Some decompressors may choose to ignore some image data to improve decompression speed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Setting Drawing Parameters” (V–2879)

Related Java Methods

`quicktime.std.image.GraphicsImporter.setQuality()`

See Also

For the corresponding get function, see `GraphicsImportGetQuality` (I–643).

GraphicsImportSetSourceRect

Sets the source rectangle to use for an imported image.

```
ComponentResult GraphicsImportSetSourceRect (
    GraphicsImportComponent ci,
    const Rect             *sourceRect );
```

ci

The component instance that identifies your connection to the graphics importer component.

sourceRect

A pointer to a Rect (IV–2399) structure defining the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the source image. Set to NIL to use the entire image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function provides a way to use only a portion of the source image.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Setting Drawing Parameters” (V–2879)

Related Java Methods

`quicktime.std.image.GraphicsImporter.setSourceRect()`

See Also

For the corresponding get function, see `GraphicsImportGetSourceRect` (I–645).

GraphicsImportValidate

Validates image data for a data reference to an imported image.

```
ComponentResult GraphicsImportValidate (
    GraphicsImportComponent  ci,
    Boolean                  *valid );
```

ci

The component instance that identifies your connection to the graphics importer component.

valid

Pointer to a Boolean value. On return, this parameter is set to `TRUE` if the the graphics importer component can draw the data reference. If the graphics importer component cannot draw the data reference, this parameter is set to `FALSE`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. Not

HasMovieChanged

all graphics importer components implement this function. A component that does not implement the function will return the `badComponentSelector` result code. This does not indicate that the file is valid or invalid.

Discussion

This function allows a graphics importer component to determine if its current data reference contains valid image data. For example, a JFIF graphics importer component might check for the presence of a JFIF marker at the start of the data stream. This function is provided for applications to use to determine what type of image data a particular file may contain. Sometimes a file may not have the correct file type or file extension. In this case, the application will not know which graphics importer component to use. By iterating through all graphics importer components and calling `GraphicsImportValidate` for each one, it may be possible to locate a graphics importer component that can draw the specified file.

Special Considerations

`GraphicsImportValidate` does not perform an exhaustive check on the file. It is possible for `GraphicsImportValidate` to claim a data reference is valid but for `GraphicsImportDraw` (I–616) to return an error due to bad data. Format-specific importers that implement the `GraphicsImportValidate` call should have the `canMovieImportValidateFile` bit set in the `flags` field of their `ComponentDescription` (IV–2212) structures.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Getting Image Characteristics” (V–2878)

Related Java Methods

`quicktime.std.image.GraphicsImporter.validate()`

H

HasMovieChanged

Determines whether a movie has changed and needs to be saved.

Boolean HasMovieChanged (


```
Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result Returns TRUE if the movie has changed, FALSE otherwise.

Discussion

Your application can clear the movie changed flag, indicating that the movie has not changed, by calling `ClearMovieChanged` (I–89).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Saving Movies” (V–2715)

Related Java Methods

`quicktime.std.movies.Movie.hasChanged()`

HitTestDSequenceData

Undocumented

```
OSErr HitTestDSequenceData (
    ImageSequence    seqID,
    void             *data,
    Size             dataSize,
    Point            where,
    long             *hit,
    long             hitFlags );
```

seqID

The unique sequence identifier that was returned by the `DecompressSequenceBegin` (I–238) function.

data

A pointer to data.

dataSize

The size of the data.

HitTestDSequenceData

where

A `Point` (IV-2339) structure that defines the hit location.

hit

Undocumented

hitFlags

Undocumented

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

hitFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Related Java Methods

`quicktime.std.image.DSequence.hitTestData()`

Volume II

Functions I–Q

I

ICMDecompressComplete

Signals the completion of a decompression operation.

```
void ICMDecompressComplete (
    ImageSequence          seqID,
    OSErr                  err,
    short                  flag,
    ICMCompletionProcRecordPtr completionRtn );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

err

Indicates whether the operation succeeded or failed. Set this parameter to 0 for successful operations. For failed operations, set the error code appropriate for the failure. For canceled operations (for example, when the ICM calls your component's `ImageCodecFlush` (II–708) function), set this parameter to –1.

flag

Completion flags (see below). Note that you may set more than one of these flags to 1.

completionRtn

A pointer to an `ICMCompletionProcRecord` (IV–2279) structure. That structure identifies the application's completion function and contains a reference constant associated with the frame. Your component obtains this structure as part of the `CodecDecompressParams` (IV–2190) structure provided by the Image Compression Manager at the start of the decompression operation.

flag Constants

codecCompletionSource

Your component is done with the source buffer. Set this flag to 1 when you are done with the processing associated with the source buffer.

codecCompletionDest

Your component is done with the destination buffer. Set this flag to 1 when you are done with the processing associated with the destination buffer.

ICMDecompressCompleteS

`codecCompletionDontUnshield`

Set this flag to 1 when you do not want the ICM to unshield the cursor as it normally would. Only codecs that are completely managing the cursor themselves should set this flag. See “Hardware Cursors” in *Inside Macintosh: QuickTime* (listed in the **bibliography**) for more information.

Discussion

Your component must call this function at the end of decompression operations.

Special Considerations

Prior to QuickTime 2.0, decompressor components called the application’s completion function directly. For compatibility, that method is still supported except for scheduled asynchronous decompression operations, which must use the ICMDecompressComplete call. Newer decompressors should always use ICMDecompressComplete rather than calling the completion function directly, regardless of the type of decompression operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Image Compression Manager Utility Functions” (V–2799)

See Also

See ICMDecompressCompleteS (II–672).

ICMDecompressCompleteS

Undocumented

```
OSErr ICMDecompressCompleteS (
    ImageSequence      seqID,
    OSErr              err,
    short              flag,
    ICMCompletionProcRecordPtr completionRtn );
```

`seqID`

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

`err`

Indicates whether the operation succeeded or failed. See “Error Codes” (IV–2663).

flag

Undocumented

completionRtn

A pointer to an ICMCompletionProcRecord (IV–2279) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

flag Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

ICMGetPixelFormatInfo

Retrieves pixel format information.

```
OSErr ICMGetPixelFormatInfo (
    OSType          PixelFormat,
    ICMPixelFormatInfoPtr theInfo );
```

PixelFormat

A constant that identifies the format; see “Pixel Formats” (IV–2686).

theInfo

A pointer to your ICMPixelFormatInfo (IV–2282) structure in which information is returned. You should initialize the size field of this structure with sizeof(ICMPixelFormatInfo). The function will not copy more than this number of bytes into the structure. On return, the size field contains the actual size of the data structure. If this amount is greater the size you passed in, that means you didn't retrieve all of the information.

function result Returns cDepthErr if the pixel format is not valid. For other errors, see “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

ICMSequenceGetChainMember

Related Java Methods

```
quicktime.std.image.ICMPixelFormatInfo.isValidPixelFormat(),  
quicktime.std.image.ICMPixelFormatInfo.getPixelFormatInfo()
```

See Also

For the corresponding set function, see ICMSetPixelFormatInfo (II–678).

ICMSequenceGetChainMember

Undocumented

```
OSErr ICMSequenceGetChainMember (  
    ImageSequence    seqID,  
    ImageSequence    *retSeqID,  
    long             flags );
```

seqID

The unique sequence identifier assigned by CompressSequenceBegin (I–119) or DecompressSequenceBegin (I–238).

retSeqID

Undocumented

flags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

ICMSequenceGetInfo

Gets multiprocessing properties for compression and decompression sequences.

```
OSErr ICMSequenceGetInfo (  
    ImageSequence    seqID,  
    OSType           which,
```



```
void          *data );
```

seqID

The unique sequence identifier assigned by CompressSequenceBegin (I–119) or DecompressSequenceBegin (I–238).

which

A constant (see below) that determines the property to be returned.

data

The value of the property indicated by the which parameter.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

which Constants

kICMSequenceTaskWeight

The multiprocessing task weight.

kICMSequenceTaskName

The multiprocessing task name. The task name has no performance impact, but it can be helpful while debugging.

Discussion

This function determines if ICM clients have requested that multiprocessor tasks assisting compression and decompression operations use specific task weights and task names.

Special Considerations

Apple’s multiprocessing capability supports both cooperatively scheduled tasks and preemptively scheduled tasks. The support for preemptively tasks allow applications to create symmetrically scheduled preemptive tasks that can be run on a single processor machine, and will take full advantage of multiple processors when they are installed.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: ImageCompression.h

See Also

For the corresponding set function, see ICMSequenceSetInfo (II–676).

ICMSequenceLockBits

Undocumented

ICMSequenceSetInfo

```
OSErr ICMSequenceLockBits (
    ImageSequence    seqID,
    PixMapPtr        dst,
    long              flags );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

dst

A pointer to a `PixMap` (IV–2332) structure.

flags

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

ICMSequenceSetInfo

Sets multiprocessing properties for compression and decompression sequences.

```
OSErr ICMSequenceSetInfo (
    ImageSequence    seqID,
    OSType            which,
    void              *data,
    Size              dataSize );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

which

A constant (see below) that determines the property to be set.

data

The value of the property to be set.

`dataSize`

The length in bytes of the `data` parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

which Constants

`kICMSequenceTaskWeight`

The multiprocessing task weight.

`kICMSequenceTaskName`

The multiprocessing task name. The task name has no performance impact, but it can be helpful while debugging.

Discussion

This function lets ICM clients request that multiprocessor tasks assisting compression and decompression operations use specific task weights and task names.

Special Considerations

Apple’s multiprocessing capability supports both cooperatively scheduled tasks and preemptively scheduled tasks. The support for preemptively tasks allow applications to create symmetrically scheduled preemptive tasks that can be run on a single processor machine, and will take full advantage of multiple processors when they are installed.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding get function, see `ICMSequenceGetInfo` (II–674).

ICMSequenceUnlockBits

Undocumented

```
OSErr ICMSequenceUnlockBits (
    ImageSequence  seqID,
    long           flags );
```

`seqID`

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

ICMSetPixelFormatInfo

flags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

ICMSetPixelFormatInfo

Lets you define your own pixel format.

```
OSErr ICMSetPixelFormatInfo (
    OSType          PixelFormat,
    ICMPixelFormatInfoPtr theInfo );
```

PixelFormat

A pixel format constant. See “Pixel Formats” (IV–2686).

theInfo

A pointer to an ICMPixelFormatInfo (IV–2282) structure containing a definition of the new pixel format.

function result Returns paramErr if the format is already defined. See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Related Java Methods

quicktime.std.image.ICMPixelFormatInfo.setPixelFormatInfo()

See Also

For the corresponding get function, see ICMGetPixelFormatInfo (II–673).

ICMShieldSequenceCursor

Hides the cursor during decompression operations.

```
OSErr ICMShieldSequenceCursor (
    ImageSequence    seqID );
```

seqID

The unique sequence identifier, assigned by `CompressSequenceBegin` (I-119) or `DecompressSequenceBegin` (I-238), for which to shield the cursor.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

For correct image display behavior, the cursor must be shielded (hidden) during decompression. By default, the Image Compression Manager handles the cursor for you, hiding it at the beginning of a decompression operation and revealing it at the end. With scheduled asynchronous decompression, however, the ICM cannot do as precise a job of managing the cursor, because it does not know exactly when scheduled operations actually begin and end. While the ICM can still manage the cursor, it must hide the cursor when each request is queued, rather than when the request is serviced. This may result in the cursor remaining hidden for long periods of time. To achieve better cursor behavior, you can choose to manage the cursor in your decompressor component. If you so choose, you can use this function to hide the cursor; the ICM displays the cursor when you call `ICMDecompressComplete` (II-671). In this manner, the cursor is hidden only when your component is decompressing and displaying the frame.

Special Considerations

This function may be called at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Image Compression Manager Utility Functions” (V-2799)

Related Java Methods

`quicktime.std.image.DSequence.shieldCursor()`

IDHCancelNotification

IDHCancelNotification

Undocumented

```
ComponentResult IDHCancelNotification (  
    ComponentInstance    idh  
    IDHNotificationID    notificationID );
```

idh

Undocumented

notificationID

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHCancelPendingIO

Undocumented

```
ComponentResult IDHCancelPendingIO (  
    ComponentInstance    idh  
    IDHParameterBlock    *pb );
```

idh

Undocumented

pb

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHCloseDevice

Undocumented

```
ComponentResult IDHCloseDevice (
    ComponentInstance    idh );
```

idh

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHDisposeNotification

Undocumented

```
ComponentResult IDHDisposeNotification (
    ComponentInstance    idh
    IDHNotificationID    notificationID );
```

idh

Undocumented

notificationID

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHGetDeviceClock

Undocumented

```
ComponentResult IDHGetDeviceClock (
```

IDHGetDeviceConfiguration

```
ComponentInstance  idh  
Component          *clock );
```

idh

Undocumented

clock

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHGetDeviceConfiguration

Undocumented

```
ComponentResult IDHGetDeviceConfiguration (   
    ComponentInstance  idh  
    QTAtomSpec         *configurationID );
```

idh

Undocumented

configurationID

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHGetDeviceControl

Undocumented

```
ComponentResult IDHGetDeviceControl (   
    ComponentInstance  idh
```



```
ComponentInstance    *deviceControl );
```

idh

Undocumented

deviceControl

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHGetDeviceList

Undocumented

```
ComponentResult IDHGetDeviceList (
    ComponentInstance    idh
    QTAAtomContainer     *deviceList );
```

idh

Undocumented

deviceList

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHGetDeviceStatus

Undocumented

```
ComponentResult IDHGetDeviceStatus (
    ComponentInstance    idh
    const QTAAtomSpec    *configurationID
```

IDHNewNotification

```
IDHDeviceStatus      *status );
```

idh

Undocumented

configurationID

Undocumented

status

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHNewNotification

Undocumented

```
ComponentResult IDHNewNotification (
    ComponentInstance      idh
    IDHDeviceID            deviceID
    IDHNotificationProc    notificationProc
    void                   *userData
    IDHNotificationID      *notificationID );
```

idh

Undocumented

deviceID

Undocumented

notificationProc

Undocumented

userData

Undocumented

notificationID

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHNotifyMeWhen

Undocumented

```
ComponentResult IDHNotifyMeWhen (
    ComponentInstance    idh
    IDHNotificationID    notificationID
    IDHEvent             events );
```

idh

Undocumented

notificationID

Undocumented

events

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHOpenDevice

Undocumented

```
ComponentResult IDHOpenDevice (
    ComponentInstance    idh
    UInt32               permissions );
```

idh

Undocumented

permissions

Undocumented

IDHRead

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHRead

Undocumented

```
ComponentResult IDHRead (
    ComponentInstance    idh
    IDHParameterBlock    *pb );
```

idh

Undocumented

pb

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHReleaseBuffer

Undocumented

```
ComponentResult IDHReleaseBuffer (
    ComponentInstance    idh
    IDHParameterBlock    *pb );
```

idh

Undocumented

pb

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHSetDeviceConfiguration

Undocumented

```
ComponentResult IDHSetDeviceConfiguration (
    ComponentInstance    idh
    const QTAtomSpec     *configurationID );
```

idh

Undocumented

configurationID

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHUpdateDeviceList

Undocumented

```
ComponentResult IDHUpdateDeviceList (
    ComponentInstance    idh
    QTAtomContainer      *deviceList );
```

idh

Undocumented

deviceList

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

IDHWrite

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

IDHWrite

Undocumented

```
ComponentResult IDHWrite (
    ComponentInstance    idh
    IDHParameterBlock    *pb );
```

idh

Undocumented

pb

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: IsochronousDataHandler.h

ImageCodecBandCompress

Asks your component to compress an image or a **band** of an image.

```
ComponentResult ImageCodecBandCompress (
    ComponentInstance    ci,
    CodecCompressParams    *params );
```

ci

An image decompressor component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

params

A pointer to a CodecCompressParams (IV–2184) structure. The Image Compression Manager places the appropriate parameter information in that structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The image being compressed may be part of a sequence.

Special Considerations

Only compressors receive this request.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecBandDecompress

Asks your component to decompress a frame.

```
ComponentResult ImageCodecBandDecompress (
    ComponentInstance      ci,
    CodecDecompressParams *params );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

params

A pointer to a `CodecDecompressParams` (IV–2190) structure. The Image Compression Manager places the appropriate parameter information in that structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

For scheduled asynchronous decompression operations, the Image Compression Manager supplies a reference to an `ICMFrameTimeRecord` (IV–2281) structure in this function’s `CodecDecompressParams` (IV–2190) structure parameter. The `ICMFrameTimeRecord` structure contains time information governing the scheduled decompression operation, including the time at which the frame must be displayed. For synchronous or immediate asynchronous decompress operations, the frame time is set to `NIL`.

When your component has finished the decompression operation, it must call the completion function. In the past, for asynchronous operations, your component called that function directly. For scheduled asynchronous decompression

ImageCodecBeginBand

operations, your component should call `ICMDecompressComplete` (II–671).

If your component sets the `codecCanAsyncWhen` flag in `predecompress` but cannot support scheduled asynchronous decompression for a given frame, it must return an error code of `codecCantWhenErr`. If your component's queue is full, it should return an error code of `codecCantQueueErr`. Only decompressors receive these requests.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecBeginBand

Called before drawing a **band** or frame; it allows your image decompressor component to save information about a band before decompressing it.

```
ComponentResult ImageCodecBeginBand (
    ComponentInstance      ci,
    CodecDecompressParams  *params,
    ImageSubCodecDecompressRecord *drp,
    long                   flags );
```

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

params

A pointer to a `CodecDecompressParams` (IV–2190) structure.

drp

A pointer to an `ImageSubCodecDecompressRecord` (IV–2289) structure.

flags

Currently unused; set to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your image decompressor component receives the address of the destination pixel map in the `baseAddr` field of the `drp` parameter. This address includes an adjustment for the offset. Note that if the bit depth of the pixel map is less than 8, your image decompressor component must adjust for the bit offset.

The `codecData` field of the `drp` parameter contains a pointer to the compressed video data. The `userDecompressRecord` field of the `drp` parameter contains a pointer to storage for the decompression operation. The storage is allocated by the base image decompressor after it calls the `ImageCodecInitialize` (II–724) function. The size of the storage is determined by the `decompressRecordSize` field of the `ImageSubCodecDecompressCapabilities` (IV–2288) structure that is returned by `ImageCodecInitialize` (II–724). Your image decompressor component should use this storage to store any additional information needed about the frame in order to decompress it.

Changes your image decompressor component makes to the `ImageSubCodecDecompressRecord` or `CodecDecompressParams` structures are preserved by the base image decompressor. For example, if your component does not need to decompress all of the data, it can change the pointer to the data to be decompressed that is stored in the `codecData` field of the `ImageSubCodecDecompressRecord` structure.

Special Considerations

Your component must implement this function. Also, the base image decompressor never calls `ImageCodecBeginBand` at interrupt time. If your component supports asynchronous scheduled decompression, it may receive more than one `ImageCodecBeginBand` call before receiving an `ImageCodecDrawBand` (II–697) call.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecBusy

Lets your component report whether it is performing an asynchronous operation.

```
ComponentResult ImageCodecBusy (
    ComponentInstance ci,
    ImageSequence seq );
```

`ci`

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

ImageCodecCancelTrigger

seq

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

function result Your component should return a result code value of 1 if an asynchronous operation is in progress; it should return a result code value of 0 if the component is not performing an asynchronous operation. You can indicate an error by returning a negative result code. See “Error Codes” (IV–2663).

Special Considerations

Both compressors and decompressors may receive this request.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecCancelTrigger

Cancels an image codec’s `ImageCodecTimeTriggerProc` (III–2096) callback.

```
ComponentResult ImageCodecCancelTrigger (  
    ComponentInstance  ci );
```

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecCreateStandardParameterDialog

Creates a parameters dialog box for a specified effect.

```
ComponentResult ImageCodecCreateStandardParameterDialog (  
    ComponentInstance  ci,
```

```

QTAtomContainer      parameterDescription,
QTAtomContainer      parameters,
QTParameterDialogOptions dialogOptions,
DialogPtr            existingDialog,
short                existingUserItem,
QTParameterDialog    *createdDialog );

```

ci

An effects component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131). The dialog box that is created will allow the user to specify the parameters of this effect.

parameterDescription

The parameter description atom container for this effect. You can obtain a valid parameter description by calling `ImageCodecGetParameterList` (II–717). A parameter is optionally **tweenable** if defined as `kAtomInterpolateIsOptional` in its parameter description atom.

parameters

The atom container that will receive the user’s chosen parameter values once the dialog has been dismissed.

dialogOptions

Controls how parameters containing **tween** data are presented in the created dialog box. If `dialogOptions` contains 0, two values are collected for each parameter that can be tweened, and the usual tweening operation will be performed for the duration of the effect being controlled. For other values, see below.

existingDialog

An existing dialog box that will have the controls from the standard parameters dialog box added to it. Set this parameter to `NIL` if you want this function to create a stand-alone dialog box.

existingUserItem

The number of the user item in the existing dialog box that should be replaced with controls from the standard parameter dialog box. You should only pass a value to this parameter if the `existingDialog` parameter is not `NIL`.

createdDialog

On return, a reference to the dialog created and displayed by the function. This reference is required by several other low-level effects functions. It will contain a valid dialog identifier even if you requested that the controls from the standard parameter dialog box be incorporated into an existing dialog box.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

ImageCodecCreateStandardParameterDialog

dialogOptions Constants

pdOptionsCollectOneValue

Only one value can be entered for parameters that are optionally **tweenable**.

This means that optionally-tweened parameters will not be tweened.

pdOptionsAllowOptionalInterpolations

The user interface for optionally-tweened parameters will be constructed to take **tweened** values. Setting this flag selects the “advanced” user interface mode. If the user interface supports this mode of operation, then a tween value can be entered for this parameter when the user interface is in the mode. An example of an advanced mode is the standard parameters dialog box; if you hold down the option key while selecting an effect, any parameters that have the kAtomInterpolateIsOptional flag set will allow a tween value to be entered.

Discussion

This is a low-level function that can be used to create a standard parameter dialog box for a specified effect, allowing the user to set the parameter values for the effect. You can optionally request that the controls from the dialog box be included within a dialog box of the calling application. The following sample code shows how to create a standard parameter dialog box and add effects controls:

```
// ImageCodecCreateStandardParameterDialog coding example
// See “Discovering QuickTime,” page 303
pMovableModalDialog = GetNewDialog(kExtraDialogID, NIL, (WindowPtr)-1);
if (pMovableModalDialog != NIL) {
    ImageCodecCreateStandardParameterDialog(
        compInstance,
        qtacParameterDescription,
        qtacEffectSample,
        pdOptionsModelDialogBox,
        pMovableModalDialog,
        kExtraUserItemID,
        &lCreatedDialogID);

    ShowWindow(pMovableModalDialog);
    SelectWindow(pMovableModalDialog);
    SetPort(pMovableModalDialog);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Low-Level Effects Functions” (V–2898)

See Also

The high-level version of this function is `QTCreateStandardParameterDialog` (II–1161).

ImageCodecDismissStandardParameterDialog

Retrieves values from a standard parameter dialog box created by the low-level `ImageCodecCreateStandardParameterDialog` (II–692) function, then closes the dialog box.

```
ComponentResult ImageCodecDismissStandardParameterDialog (
    ComponentInstance ci,
    QTParameterDialog createdDialog );
```

ci

An effect component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131). This must be the instance passed to `ImageCodecCreateStandardParameterDialog` (II–692) to create the dialog box.

createdDialog

A reference to the dialog box created by the call to `ImageCodecCreateStandardParameterDialog`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function should be called after the `ImageCodecIsStandardParameterDialogEvent` (II–726) function returns `codecParameterDialogConfirm` or `userCanceledErr`, which indicate that the user has dismissed the dialog box. The function dismisses the dialog box, deallocating any memory allocated during the call to `ImageCodecCreateStandardParameterDialog` (II–692).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Low-Level Effects Functions” (V–2898)

ImageCodecDisposeImageGWorld

ImageCodecDisposeImageGWorld

Disposes of an image graphics world associated with an image codec.

```
ComponentResult ImageCodecDisposeImageGWorld (
    ComponentInstance  ci,
    GWorldPtr          theGW );
```

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

theGW

A pointer to a `CGrafPort` (IV–2168) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecDisposeMemory

Disposes codec-allocated memory.

```
ComponentResult ImageCodecDisposeMemory (
    ComponentInstance  ci,
    Ptr                data );
```

ci

An image compressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

data

A pointer to the previously allocated memory block.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your component receives the `ImageCodecDisposeMemory` request whenever an application calls `CDSequenceDisposeMemory` (I–77).

Special Considerations

When a codec instance is closed, it must ensure that all blocks allocated by that instance are disposed and call `ICMMemoryDisposedProc` (III–2092).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecDrawBand

Decompresses a **band** or frame.

```
ComponentResult ImageCodecDrawBand (
    ComponentInstance      ci,
    ImageSubCodecDecompressRecord *drp );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

drp

A pointer to an `ImageSubCodecDecompressRecord` (IV–2289) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

When the base image decompressor calls your image decompressor component’s `ImageCodecDrawBand` function, your component must perform the decompression specified by the fields of the `ImageSubCodecDecompressRecord` (IV–2289) structure. The structure includes any changes your component made to it when performing the `ImageCodecBeginBand` (II–690) function. If your component supports asynchronous scheduled decompression, it may receive more than one `ImageCodecBeginBand` call before receiving an `ImageCodecDrawBand` call.

Special Considerations

Your component must implement this function. If the `ImageSubCodecDecompressRecord` structure specifies a **progress function** or data-loading function, the base image decompressor never calls this function at interrupt time. If not, the base image decompressor may call this function at interrupt time.

ImageCodecDroppingFrame

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecDroppingFrame

Undocumented

```
ComponentResult ImageCodecDroppingFrame (
    ComponentInstance      ci,
    const ImageSubCodecDecompressRecord *drp );
```

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

drp

A pointer to an `ImageSubCodecDecompressRecord` (IV–2289) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

ImageCodecEffectBegin

Undocumented

```
ComponentResult ImageCodecEffectBegin (
    ComponentInstance      effect,
    CodecDecompressParams  *p,
    EffectsFrameParamsPtr  ePtr );
```

effect

An effect component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

p

A pointer to a CodecDecompressParams (IV–2190) structure.

ePtr

A pointer to a EffectsFrameParams (IV–2242) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.**Version Notes**

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

ImageCodecEffectCancel

Undocumented

```
ComponentResult ImageCodecEffectCancel (
    ComponentInstance    effect,
    EffectsFrameParamsPtr p );
```

effect

An effect component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

p

A pointer to a EffectsFrameParams (IV–2242) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.**Version Notes**

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

ImageCodecEffectConvertEffectSourceToFormat

Undocumented

```
ComponentResult ImageCodecEffectConvertEffectSourceToFormat (
    ComponentInstance    effect,
    EffectSourcePtr      sourceToConvert,
    ImageDescriptionHandle requestedDesc );
```

ImageCodecEffectDisposeSMPTEFrame

effect

An effect component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

sourceToConvert

Undocumented

requestedDesc

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecEffectDisposeSMPTEFrame

Undocumented

```
ComponentResult ImageCodecEffectDisposeSMPTEFrame (  
    ComponentInstance    effect,  
    SMPTEFrameReference  frameRef );
```

effect

An effect component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

frameRef

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecEffectGetSpeed

Undocumented

```
ComponentResult ImageCodecEffectGetSpeed (
    ComponentInstance    effect,
    QTAtomContainer      parameters,
    Fixed                *pFPS );
```

effect

An effect component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

parameters

Undocumented

pFPS

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

ImageCodecEffectPrepareSMPTEFrame

Undocumented

```
ComponentResult ImageCodecEffectPrepareSMPTEFrame (
    ComponentInstance    effect,
    PixMapPtr            destPixMap,
    SMPTEFrameReference  *returnValue );
```

effect

An effect component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

destPixMap

Undocumented

returnValue

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

ImageCodecEffectRenderFrame

Programming Info

C interface file: ImageCodec.h

ImageCodecEffectRenderFrame

Undocumented

```
ComponentResult ImageCodecEffectRenderFrame (  
    ComponentInstance      effect,  
    EffectsFrameParamsPtr  p );
```

effect

An effect component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

p

A pointer to an `EffectsFrameParams` (IV–2242) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

ImageCodecEffectRenderSMPTEFrame

Undocumented

```
ComponentResult ImageCodecEffectRenderSMPTEFrame (  
    ComponentInstance      effect,  
    PixMapPtr              destPixMap,  
    SMPTEFrameReference    frameRef,  
    Fixed                  effectPercentageEven,  
    Fixed                  effectPercentageOdd,  
    Rect                   *pSourceRect,  
    MatrixRecord            *pMatrix,  
    SMPTEWipeType          effectNumber,  
    long                   xRepeat,  
    long                   yRepeat,  
    SMPTEFlags              flags,  
    Fixed                  penWidth,  
    long                   strokeValue );
```

effect

An effect component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

destPixMap

Undocumented

frameRef

Undocumented

effectPercentageEven

Undocumented

effectPercentageOdd

Undocumented

pSourceRect

Undocumented

pMatrix

Undocumented

effectNumber

Undocumented

xRepeat

Undocumented

yRepeat

Undocumented

flags

Undocumented

penWidth

Undocumented

strokeValue

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecEffectSetup

ImageCodecEffectSetup

Undocumented

```
ComponentResult ImageCodecEffectSetup (
    ComponentInstance    effect,
    CodecDecompressParams *p );
```

effect

An effect component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

p

A pointer to a `CodecDecompressParams` (IV–2190) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecEndBand

Notifies your image decompressor component that decompression of a **band** has finished or that it was terminated by the Image Compression Manager.

```
ComponentResult ImageCodecEndBand (
    ComponentInstance    ci,
    ImageSubCodecDecompressRecord *drp,
    OSErr                result,
    long                  flags );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

drp

A pointer to an `ImageSubCodecDecompressRecord` (IV–2289) structure.

result

A result code.

flags

Currently unused; set to 0.

function result Returns noErr if the **band** or frame was drawn successfully. If it is any other value, the band or frame was not drawn. See “Error Codes” (IV–2663).

Discussion

Your image decompressor component is not required to implement this function. After your image decompressor component handles a call to this function, it can perform any tasks that are required when decompression is finished, such as disposing of data structures that are no longer needed. Because this function can be called at interrupt time, your component cannot use it to dispose of data structures; this must occur after handling the function.

Special Considerations

The base image decompressor may call ImageCodecEndBand at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecExtractAndCombineFields

Performs field operations on video data.

```
ComponentResult ImageCodecExtractAndCombineFields (
    ComponentInstance      ci,
    long                   fieldFlags,
    void                   *data1,
    long                   dataSize1,
    ImageDescriptionHandle desc1,
    void                   *data2,
    long                   dataSize2,
    ImageDescriptionHandle desc2,
    void                   *outputData,
    long                   *outDataSize,
    ImageDescriptionHandle descOut );
```

ci

An image decompressor component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

ImageCodecExtractAndCombineFields

`fieldFlags`

Flags (see below) that specify the operation to be performed. A correctly formed request will specify two input fields, mapping one to the odd output field and the other to the even output field.

`data1`

A pointer to a buffer containing the compressed image data for the first input field.

`dataSize1`

The size of the `data1` buffer.

`desc1`

An `ImageDescription` (IV–2285) structure describing the format and characteristics of the data in the `data1` buffer.

`data2`

A pointer to a buffer containing the compressed image data for the second input field. Set to `NIL` if the requested operation uses only one input frame.

`dataSize2`

The size of the `data2` buffer. Set to 0 if the requested operation uses only one input frame.

`desc2`

An `ImageDescription` (IV–2285) structure describing the format and characteristics of the data in the `data2` buffer. Set to `NIL` if the requested operation uses only one input frame.

`outputData`

A pointer to a buffer to receive the resulting frame.

`outDataSize`

On output this parameter returns the actual size of the new compressed image data.

`descOut`

The desired format of the resulting frames. Typically this is the same format specified by the `desc1` and `desc2` parameters.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

fieldFlags Constants

`evenField1ToEvenFieldOut`

Maps the even field specified by the `data1` parameter to the even output field.

`evenField1ToOddFieldOut`

Maps the even field specified by the `data1` parameter to the odd output field.

`oddField1ToEvenFieldOut`

Maps the odd field specified by the `data1` parameter to the even output field.

`oddField1ToOddFieldOut`

Maps the odd field specified by the `data1` parameter to the odd output field.

`evenField2ToEvenFieldOut`

Maps the even field specified by the `data2` parameter to the even output field.

`evenField2ToOddFieldOut`

Maps the even field specified by the `data2` parameter to the odd output field.

`oddField2ToEvenFieldOut`

Maps the odd field specified by the `data2` parameter to the even output field.

`oddField2ToOddFieldOut`

Maps the odd field specified by the `data2` parameter to the odd output field.

Discussion

This function allows fields from two separate images, compressed in the same format, to be combined into a new compressed frame. Typically the operation results in an image of identical quality to the source images. Fields of a single image may also be duplicated or reversed by this function. The component selector for this function is:

`kImageCodecExtractAndCombineFieldsSelect` = 0x0015

Special Considerations

This codec routine implements the functionality of the `ImageFieldSequenceExtractCombine` (II-747) function. If your codec is capable of separately compressing both fields of a video frame, you should incorporate support for this function. Your codec must ensure that it understands the image formats specified by `desc1` and `desc2` parameters, as these may not be the same as the codec's native image format. The image format specified by the `descOut` parameter will be the same as the codec's native image format.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: "Base Image Decompressor Functions" (V-2805)

See Also

See `ImageFieldSequenceExtractCombine` (II-747).

ImageCodecFlush

ImageCodecFlush

Empties an image decompressor component's input queue of pending scheduled frames.

```
ComponentResult ImageCodecFlush (  
    ComponentInstance ci );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your component receives the `ImageCodecFlush` function whenever the Image Compression Manager needs to cancel the display of all scheduled frames. Your decompressor should empty its queue of scheduled asynchronous decompression requests. For each request, your component must call `ICMDecompressComplete` (II–671). Be sure to set the `err` parameter to `-1`, indicating that the request was canceled. Also, you must set both the `codecCompletionSource` and `codecCompletionDest` flags to 1.

Special Considerations

Your component's `ImageCodecFlush` function may be called at interrupt time. Only decompressor components that support scheduled asynchronous decompression receive this call.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecFlushFrame

Undocumented

```
ComponentResult ImageCodecFlushFrame (  
    ComponentInstance ci,  
    UInt32 flags );
```

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

flags

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecGetBaseMPWorkFunction

Gets an image codec’s `ComponentMPWorkFunctionProc` (III–2077) callback.

```
ComponentResult ImageCodecGetBaseMPWorkFunction (
    ComponentInstance          ci,
    ComponentMPWorkFunctionUPP *workFunction,
    void                      **refCon,
    ImageCodecMPDrawBandUPP   drawProc,
    void                      *drawProcRefCon );
```

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

workFunction

On return, a pointer to a `ComponentMPWorkFunctionProc` (III–2077) callback.

refCon

On return, a handle to the reference constant that is passed to the `ComponentMPWorkFunctionProc` callback.

drawProc

An `ImageCodecMPDrawBandProc` (III–2095) callback.

ImageCodecGetCodecInfo

`drawProcRefCon`

A pointer to a reference constant that is passed to your `ImageCodecMPDrawBandProc` callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecGetCodecInfo

Notifies your codec whenever an application calls `GetCodecInfo` (I–385).

```
ComponentResult ImageCodecGetCodecInfo (  
    ComponentInstance ci,  
    CodecInfo *info );
```

`ci`

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`info`

A pointer to a `CodecInfo` (IV–2202) structure to update. Your component should report its capabilities by formatting the structure in the location specified by this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Both compressors and decompressors may receive this request.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecGetCompressedImageSize

Notifies your codec whenever an application calls `GetCompressedImageSize` (I–396).

```
ComponentResult ImageCodecGetCompressedImageSize (
    ComponentInstance      ci,
    ImageDescriptionHandle desc,
    Ptr                   data,
    long                  bufferSize,
    ICMDDataProcRecordPtr dataProc,
    long                  *dataSize );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

desc

A handle to the `ImageDescription` (IV–2285) structure that defines the compressed image for the operation.

data

A pointer to the compressed image data.

bufferSize

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If the application did not specify a data-loading function this parameter is NIL.

dataProc

A pointer to an `ICMDDataProcRecord` structure. If the data stream is not all in memory when the application calls `GetCompressedImageSize` (I–396), your component may call an application function that loads more compressed data.

dataSize

A pointer to a field that is to receive the size, in bytes, of the compressed image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

Only decompressors receive this request.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecGetCompressionTime

Notifies your codec whenever an application calls `GetCompressionTime` (I–401).

ImageCodecGetCompressionTime

```
ComponentResult ImageCodecGetCompressionTime (
    ComponentInstance ci,
    PixMapHandle src,
    const Rect *srcRect,
    short depth,
    CodecQ *spatialQuality,
    CodecQ *temporalQuality,
    unsigned long *time );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

src

A handle to the source image. The source image is stored in a `PixMap` (IV–2332) structure. Applications may use the time information you return for more than one image. Consequently, your compressor should not consider the contents of the image when determining the maximum compression time. Rather, you should consider only the quality level, pixel depth, and image size. This parameter may also be set to `NIL`. In this case the application has not supplied a source image; your component should use the other parameters to determine the characteristics of the image to be compressed.

srcRect

A pointer to a `Rect` (IV–2399) structure that defines the portion of the source image to compress.

depth

The depth at which the image is to be compressed. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 indicate 1-bit, 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

spatialQuality

A pointer to a field containing the desired compressed image quality (see below). The compressor sets this field to the closest actual quality that it can achieve.

temporalQuality

A pointer to a field containing the desired sequence temporal quality (see below). The compressor sets this field to the closest actual quality that it can achieve.

time

A pointer to a field to receive the compression time, in milliseconds. If your component cannot determine the amount of time required to compress the

image, set this field to 0. Check to see if the value of this field is `NIL` and, if so, do not write to location 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

spatialQuality and temporalQuality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecGetDecompressLatency

Retrieves the video latency value from a specified video codec.

```
ComponentResult ImageCodecGetDecompressLatency (
    ComponentInstance ci,
    TimeRecord        *latency );
```

`ci`

A video codec component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

ImageCodecGetMaxCompressionSize

latency

Pointer to a `TimeRecord` (IV–2486) structure containing the latency time required for that codec.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecGetMaxCompressionSize

Notifies your codec whenever an application calls `GetMaxCompressionSize` (I–420).

```
ComponentResult ImageCodecGetMaxCompressionSize (
    ComponentInstance ci,
    PixMapHandle src,
    const Rect *srcRect,
    short depth,
    CodecQ quality,
    long *size );
```

`ci`

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`src`

A handle to the source image. The source image is stored in a `PixMap` (IV–2332) structure. Applications use the size information you return to allocate buffers that may be used for more than one image. Consequently, your compressor should not consider the contents of the image when determining the maximum compressed size. Rather, you should consider only the quality level, pixel depth, and image size. This parameter may also be set to `NIL`. In this case the application has not supplied a source image; your component should use the other parameters to determine the characteristics of the image to be compressed.

`srcRect`

A pointer to a `Rect` (IV–2399) structure that defines the portion of the source image to compress.

depth

The depth at which the image is to be compressed. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 indicate 1-bit, 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

quality

The desired compressed image quality (see below).

size

A pointer to a field to receive the maximum size, in bytes, of the compressed image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

quality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

The caller uses this function to determine the maximum size the data will become for a given parameter. Your component returns a long integer indicating the maximum number of bytes of compressed data that results from compressing the specified image.

Special Considerations

Only compressors receive this request.

Version Notes

Introduced in QuickTime 3 or earlier.

ImageCodecGetMaxCompressionSizeWithSources

Programming Info

C interface file: ImageCodec.h

ImageCodecGetMaxCompressionSizeWithSources

Notifies your codec when an application calls `GetCSequenceMaxCompressionSize` (I–405).

```
ComponentResult ImageCodecGetMaxCompressionSizeWithSources (
    ComponentInstance      ci,
    PixMapHandle           src,
    const Rect             *srcRect,
    short                  depth,
    CodecQ                 quality,
    CDSequenceDataSourcePtr sourceData,
    long                   *size );
```

`ci`

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`src`

A handle to the source image. The source image is stored in a `PixMap` (IV–2332) structure. Applications use the size information you return to allocate buffers for more than one image. Consequently, your compressor should not consider the contents of the image when determining the maximum compressed size. Rather, you should consider only the quality level, pixel depth, and image size. This parameter may also be set to `NIL`. In this case the application has not supplied a source image; your component should use the other parameters to determine the characteristics of the image to be compressed.

`srcRect`

A pointer to a `Rect` (IV–2399) structure that defines the portion of the source image to compress.

`depth`

The depth at which the image is to be compressed. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 indicate 1-bit, 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

`quality`

The desired compression image quality.

sourceData

A pointer to a CDSequenceDataSource (IV–2165) structure, which contains a linked list of all data sources. Because each data source contains a link to the next data source, a codec can access all data sources from this structure.

size

A pointer to a field to receive the maximum size, in bytes, of the compressed image.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The caller uses this function to determine the maximum size the data will be compressed to for a given image and set of data sources.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Base Image Decompressor Functions” (V–2805)

See Also

This function is similar in purpose to ImageCodecGetMaxCompressionSize (II–714). However, it also considers data sources that the codec may compress with the image.

ImageCodecGetParameterList

Returns a parameter description atom container for a specified effect component instance.

```
ComponentResult ImageCodecGetParameterList (
    ComponentInstance ci,
    QTAtomContainer *parameterDescriptionHandle );
```

ci

An effect component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

parameterDescription

The returned atom container for this component instance.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

ImageCodecGetParameterListHandle

Discussion

This function returns the parameter description for the effect specified by the component instance *ci*, as a handle containing an 'atms' **resource** of ID 1. The handle should be detached if it has been read in from a resource. Each parameter of the effect is described in the parameter description, with details of its name, type, legal values and hints about how a user interface to the parameter should be constructed.

Special Considerations

The calling application is responsible for disposing of the QT atom container returned in *parameterDescription*. The application should do this by calling *QTDisposeAtomContainer* (II–1164) once it has finished using the parameter description.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: *ImageCodec.h*

Programming summary: “Low-Level Effects Functions” (V–2898)

ImageCodecGetParameterListHandle

Returns a handle to a Mac OS **resource** of type 'atms'.

```
ComponentResult ImageCodecGetParameterListHandle (
    ComponentInstance  ci,
    Handle             *parameterDescriptionHandle );
```

ci

An effect component instance. Your software obtains this reference from *OpenComponent* (II–1130) or *OpenDefaultComponent* (II–1131).

parameterDescriptionHandle

A pointer to a handle to a Mac OS **resource** of type 'atms'.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Discussion

The purpose of this function is to build a QT atom container in response to a call to *ImageCodecGetParameterList* (II–717). The handle should be detached if it has been read in from a **resource**. The caller is responsible for disposing of the handle.

Special Considerations

The implementation of this function in the Base Effect component reads in and detaches a Component Public Resource of type 'atms' and ID 1.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

ImageCodecGetSettings

Returns the codec settings chosen by the user.

```
ComponentResult ImageCodecGetSettings (
    ComponentInstance ci,
    Handle settings );
```

ci

An image compressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

settings

A handle that the codec should resize and fill in with the current internal settings. If there are no current internal settings, resize it to 0. Don't dispose of this handle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

`ImageCodecGetSettings` returns the codec's current internal settings. If there are no current settings or the settings are the same as the defaults, the codec can set the handle to `NIL`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Base Image Decompressor Functions” (V–2805)

See Also

For the corresponding set function, see `ImageCodecSetSettings` (II–737).

ImageCodecGetSettingsAsText

ImageCodecGetSettingsAsText

Undocumented

```
ComponentResult ImageCodecGetSettingsAsText (  
    ComponentInstance ci,  
    Handle            *text );
```

ci

An image compressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

text

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecGetSimilarity

Notifies your codec when an application calls `GetSimilarity` (I–508).

```
ComponentResult ImageCodecGetSimilarity (  
    ComponentInstance ci,  
    PixMapHandle      src,  
    const Rect        *srcRect,  
    ImageDescriptionHandle desc,  
    Ptr               data,  
    Fixed              *similarity );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

src

A handle to the noncompressed image. The image is stored in a `PixMap` (IV–2332) structure.

srcRect

A pointer to a `Rect` (IV–2399) structure that defines the portion of the image to compare to the compressed image.

desc

A handle to the `ImageDescription` (IV–2285) structure that defines the compressed image for the operation.

data

A pointer to the compressed image data.

similarity

A pointer to a field that is to receive the similarity value. Your component sets this field to reflect the relative similarity of the two images. Valid values range from 0 (key frame) to 1 (identical).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Only decompressors receive this request.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecGetSourceDataGammaLevel

Returns the native gamma of compressed data, if any.

```
ComponentResult ImageCodecGetSourceDataGammaLevel (
    ComponentInstance ci,
    Fixed *sourceDataGammaLevel );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

sourceDataGammaLevel

On return, the native gamma level of the compressed data. If the value is 0, it is the default gamma level of the platform.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The ICM uses the information returned by this function to determine what gamma correction is necessary. For example, the Apple DV Codec returns 2.2.

Version Notes

Introduced in QuickTime 5.

ImageCodecHitTestData

Programming Info

C interface file: ImageCodec.h

See Also

See ImageCodecRequestGammaLevel (II-734).

ImageCodecHitTestData

Notifies your codec when the application calls PtInDSequenceData (II-1147).

```
ComponentResult ImageCodecHitTestData (
    ComponentInstance      ci,
    ImageDescriptionHandle desc,
    void                   *data,
    Size                   dataSize,
    Point                  where,
    Boolean                 *hit );
```

ci

An image decompressor component instance. Your software obtains this reference from OpenComponent (II-1130) or OpenDefaultComponent (II-1131).

desc

An ImageDescriptionHandle for the image data pointed to by the data parameter.

data

Pointer to compressed data in the format specified by the desc parameter.

dataSize

Size of the compressed data referred to by the data parameter.

where

A QuickDraw Point (IV-2339) structure (0,0) based at the top-left corner of the image.

hit

A pointer to a Boolean value. The value should be set to TRUE if the point specified by the where parameter is contained within the compressed image data specified by the data parameter, or FALSE if the specified point falls within a blank portion of the image.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Discussion

ImageCodecHitTestData allows the calling application to perform hit testing on compressed data.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecHitTestDataWithFlags*Undocumented*

```
ComponentResult ImageCodecHitTestDataWithFlags (
    ComponentInstance      ci,
    ImageDescriptionHandle desc,
    void                  *data,
    Size                   dataSize,
    Point                  where,
    long                   *hit,
    long                   hitFlags );
```

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

desc

A handle to an `ImageDescription` (IV–2285) structure for the image data pointed to by the *data* parameter.

data

Pointer to compressed data in the format specified by the *desc* parameter.

dataSize

Size of the compressed data referred to by the *data* parameter.

where

A `QuickDraw Point` (IV–2339) structure (0,0) based at the top-left corner of the image.

hit

A pointer to a `Boolean`. The `Boolean` should be set to `TRUE` if the point specified by the *where* parameter is contained within the compressed image data specified by the *data* parameter.

hitFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

ImageCodecInitialize

hitFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

ImageCodecInitialize

Called before making any other all calls to your component.

```
ComponentResult ImageCodecInitialize (
    ComponentInstance      ci,
    ImageSubCodecDecompressCapabilities *cap );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

cap

On return, an `ImageSubCodecDecompressCapabilities` (IV–2288) structure that contains the capabilities of the image decompressor component. This structure contains two fields. The `canAsync` field specifies whether your component can support asynchronous decompression operations. The `decompressRecordSize` field specifies the size of the decompression record structure for your component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your component must implement this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecIsImageDescriptionEquivalent

Compares image descriptions.

```
ComponentResult ImageCodecIsImageDescriptionEquivalent (
    ComponentInstance      ci,
    ImageDescriptionHandle newDesc,
    Boolean                *equivalent );
```

ci

An image compressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

newDesc

A handle to the `ImageDescription` (IV–2285) structure that describes the compressed image.

equivalent

A pointer to a Boolean value. If the structure provided in the `newDesc` parameter is equivalent to the `ImageDescription` (IV–2285) structure currently in use by the image sequence, this value is set to `TRUE`. If they are not equivalent, and therefore a new image sequence must be created to display an image using the new `ImageDescription` structure, this value is set to `FALSE`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your component receives this call whenever an application calls `CDSequenceEquivalentImageDescription` (I–78). Implementing this function can significantly improve playback of edited video sequences using your codec. For example, if two sequences are compressed at different quality levels and are edited together they will have different image descriptions because their quality values will be different. This will force QuickTime to use two separate decompressor instances to display the images. By implementing this function your decompressor can tell QuickTime that differences in quality levels don’t require separate decompressors. This saves memory and time, thus improving performance.

Special Considerations

The current `ImageDescription` (IV–2285) structure is not passed in this function because the Image Compression Manager assumes the codec has already made copies of all relevant data fields from the current `ImageDescription` structure during the `ImageCodecPreDecompress` (II–731) call.

Version Notes

Introduced in QuickTime 3 or earlier.

ImageCodecIsStandardParameterDialogEvent

Programming Info

C interface file: ImageCodec.h

Programming summary: “Base Image Decompressor Functions” (V–2805)

See Also

See CDSequenceEquivalentImageDescription (I–78).

ImageCodecIsStandardParameterDialogEvent

Processes events related to a standard parameters dialog box created by ImageCodecCreateStandardParameterDialog (II–692).

```
ComponentResult ImageCodecIsStandardParameterDialogEvent (
    ComponentInstance ci,
    EventRecord *pEvent,
    QTPParameterDialog createdDialog );
```

ci

An effect component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131). This must be the instance that was passed to ImageCodecCreateStandardParameterDialog (II–692) to create the dialog box.

pEvent

A pointer to an EventRecord (IV–2246) structure.

createdDialog

A reference to the dialog box created by the call to ImageCodecCreateStandardParameterDialog (II–692).

function result If the error code returned is featureUnsupported, your application should process the event in the normal way. If it is noErr, the event was processed. If this function returns any other value, an error occurred. See “Error Codes” (IV–2663).

Discussion

This function returns an error code that indicates whether the event pointed to by pEvent was processed or not. After you call ImageCodecCreateStandardParameterDialog (II–692) to create a standard parameter dialog box, you must pass every non-null event to this function. It processes events related to the standard parameter dialog box, passing other events to your application for processing.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Low-Level Effects Functions” (V–2898)

See Also

The high-level version of this function is `QTIsStandardParameterDialogEvent` (II–1186). Use the high-level version to handle events for dialog boxes created by `QTCreateStandardParameterDialog` (II–1161).

ImageCodecNewImageBufferMemory

Asks a codec to allocate memory for an offscreen buffer of non-RGB pixels.

```
ComponentResult ImageCodecNewImageBufferMemory (
    ComponentInstance ci,
    CodecDecompressParams *params,
    long flags,
    ICMemoryDisposedUPP memoryGoneProc,
    void *refCon );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

params

A pointer to a decompression parameters structure.

flags

Currently, this parameter is always set to 0.

memoryGoneProc

A pointer to an `ICMemoryDisposedProc` (III–2092) callback that will be called before disposing of the memory allocated by the codec.

refCon

A reference constant that is passed to your `ICMemoryDisposedProc` callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This call is used to support a codec decompressing into a non-RGB buffer. The transfer codec is responsible for defining the offscreen buffer and transferring the image from the offscreen buffer to the destination. Your component receives this

ImageCodecNewImageGWorld

call whenever another codec has requested a non-RGB offscreen buffer of the type of your component's subtype.

Special Considerations

The Image Compression Manager does not currently track memory allocations. When a compressor or decompressor component instance is closed, it must ensure that all blocks allocated by that instance are disposed of and call the `ICMemoryDisposedProc` (III–2092) callback.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecNewImageGWorld

Undocumented

```
ComponentResult ImageCodecNewImageGWorld (
    ComponentInstance      ci,
    CodecDecompressParams *params,
    GWorldPtr              *newGW,
    long                   flags );
```

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

params

A pointer to a `CodecDecompressParams` (IV–2190) structure.

newGW

A pointer to a pointer to a `CGrafPort` (IV–2168) structure.

flags

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

ImageCodecNewMemory

Requests codec-allocated memory.

```
ComponentResult ImageCodecNewMemory (
    ComponentInstance    ci,
    Ptr                  *data,
    Size                  dataSize,
    long                  dataUse,
    ICMemoryDisposedUPP memoryGoneProc,
    void                  *refCon );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

data

Returns a pointer to the allocated memory.

dataSize

The desired size of the data buffer.

dataUse

A constant (see below) that indicates how the memory is to be used. For example, the memory may be used to store compressed data before it's displayed, mask plane data, or decompressed data. If there is no benefit to storing a particular kind of data in codec memory, the codec should refuse the request for the memory allocation.

memoryGoneProc

A pointer to an `ICMemoryDisposedProc` (III–2092) callback that will be called before disposing of the memory allocated by a codec.

refCon

A reference constant value that your codec must pass to the `memoryGoneProc` callback. Use this parameter to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. If your codec does not currently have free memory for compression frame data, but will soon, you can return `codecNoMemoryPleaseWaitErr` to indicate this fact.

ImageCodecPreCompress

dataUse Constants

0x00000001

Memory will be used for holding compressed image data.

0x00000002

Memory will be used for an offscreen image buffer.

Discussion

Some hardware codecs may have on-board memory that can be used to store compressed and /or decompressed data. This function makes this memory available for use by clients of the codec. Some software codecs may be able to optimize their performance by having more control over memory allocation. This function makes such control available. Your component receives this call whenever an application calls `CDSequenceNewMemory` (I–84).

Special Considerations

The Image Compression Manager does not currently track memory allocations. When a compressor or decompressor component instance is closed, it must ensure that all blocks allocated by that instance are disposed and call the `ICMemoryDisposedProc` (III–2092) callback.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecPreCompress

Notifies your component before compressing an image or a **band** of an image.

```
ComponentResult ImageCodecPreCompress (  
    ComponentInstance    ci,  
    CodecCompressParams  *params );
```

`ci`

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`params`

A pointer to a `CodecCompressParams` (IV–2184) structure. The Image Compression Manager places the appropriate parameter information in that structure.

function result See “Error Codes” (IV–2663). Your component should return a result code of `codecConditionErr` if it cannot field the compression request. Return `noErr` if there is no error.

Discussion

Your component receives this call before compressing an image or a **band** of an image. The Image Compression Manager also calls this function when processing a sequence. In that case, the Image Compression Manager calls this function whenever the parameters governing the sequence operation have changed substantially. Your component indicates whether it can perform the requested compression operation.

Special Considerations

Only compressors receive this call.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecPreDecompress

Notifies your component before decompressing an image or sequence of frames.

```
ComponentResult ImageCodecPreDecompress (
    ComponentInstance      ci,
    CodecDecompressParams *params );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

params

A pointer to a `CodecDecompressParams` (IV–2190) structure. The Image Compression Manager places the appropriate parameter information in that structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If your decompressor component supports scheduled asynchronous decompression operations, be sure to set the `codecCanAsyncWhen` flag to 1 in the `flags` field of your component’s `CodecCapabilities` (IV–2179) structure. If you set `codecCanAsyncWhen`, you must also set `codecCanAsync`. Codecs that support scheduled

ImageCodecPreflight

asynchronous decompression are strongly advised to also set the `codecCanShieldCursor` flag.

If your decompressor component uses a secondary hardware buffer for its images, be sure to set the `codecHasVolatileBuffer` flag to 1 in the `flags` field of your component's `CodecCapabilities` structure. If your decompressor component is used solely as a transfer codec and uses the `ImageCodecNewImageBufferMemory` (II-727) function to create an offscreen buffer that is really onscreen, your codec will need to set the `codecImageBufferIsOnScreen` flag to 1.

Special Considerations

Only decompressors receive this request.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecPreflight

Called before decompressing an image, in response to an `ImageCodecPreDecompress` (II-731) call from the Image Compression Manager.

```
ComponentResult ImageCodecPreflight (
    ComponentInstance      ci,
    CodecDecompressParams  *params );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131).

params

A pointer to a `CodecDecompressParams` (IV-2190) structure.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

Your codec responds to this call by returning information about its capabilities in a `CodecCapabilities` (IV-2179) structure. The Image Compression Manager creates the decompression parameters structure, and your image decompressor component is required only to provide values for the `wantedDestinationPixelSize` and `wantedDestinationPixelTypes` fields of the structure. Your image decompressor component can also modify other fields if necessary. For example, if it can scale

images, it must set the `codecCapabilityCanScale` flag in the `capabilities` field of the structure.

Special Considerations

Your component must implement this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecQueueStarting

Called by the base image decompressor before decompressing the frames in the queue if your image decompressor component supports asynchronous scheduled decompression.

```
ComponentResult ImageCodecQueueStarting (
    ComponentInstance ci );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your component is not required to implement this function. It can perform any tasks at this time, such as locking data structures.

Special Considerations

The base image decompressor never calls this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecQueueStopping

ImageCodecQueueStopping

Notifies your component that the frames in the queue have been decompressed, if your image decompressor component supports asynchronous scheduled decompression.

```
ComponentResult ImageCodecQueueStopping (  
    ComponentInstance    ci );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

After your image decompressor component handles a call to this function, it can perform any tasks that are required when decompression of the frames is finished, such as disposing of data structures that are no longer needed.

Special Considerations

Your component is not required to implement this function. This function is never called at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecRequestGammaLevel

Asks an image codec to convert from source to destination gamma levels.

```
ComponentResult ImageCodecRequestGammaLevel (  
    ComponentInstance    ci,  
    Fixed                srcGammaLevel,  
    Fixed                dstGammaLevel,  
    long                 *codecCanMatch );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`srcGammaLevel`
The gamma level to convert from.

`dstGammaLevel`
The gamma level to convert to.

`codecCanMatch`
Pointer to a value that indicates if the conversion from `srcGammaLevel` to `dstGammaLevel` is supported.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function tells the codec what the gamma of the source buffer and destination pixel map are so that the codec can try to convert between the two gammas when decompressing. Proper gamma conversion is accomplished by normalizing source data to black and white points to 0 to 1 and raising the result by the ratio of the `srcGammaLevel` divided by `dstGammaLevel`. The most accurate correction is done in RGB space, but a visual approximation can be done by raising the luma component alone.

Special Considerations

This function can be called several times as the ICM sets up a gamma conversion chain. The last value takes precedent for future scheduled frames. It may also be called while frames are already scheduled, indicating that conditions have changed. The new request is effective on frames that are scheduled after the call is made. Frames previously scheduled should continue to use the previously requested gamma conversion values.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCodec.h`

See Also

See `ImageCodecGetSourceDataGammaLevel` (II–721).

ImageCodecRequestSettings

Displays a dialog box containing codec-specific compression settings.

```
ComponentResult ImageCodecRequestSettings (
    ComponentInstance ci,
    Handle settings,
    Rect *rp,
```

ImageCodecScheduleFrame

```
ModalFilterUPP      filterProc );
```

ci

An image compressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

settings

A handle of data specific to the codec. If the handle is empty, the codec should use its default settings.

rp

A pointer to a `Rect` (IV–2399) structure giving the coordinates of the standard compression dialog box in global screen coordinates. The codec can use this to position its dialog box in the same area of the screen.

filterProc

A pointer to a `ModalFilterProc` (III–2098) callback that the codec must either pass to the Mac OS `ModalDialog` function or call at the beginning of the codec dialog process. This callback gives the calling application and standard compression dialog box a chance to process update events.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The `ImageCodecRequestSettings` function allows the display of a dialog box of additional compression settings specific to the codec. These settings are stored in a settings handle. The codec can store any data in any format it wants in the settings handle and resize it accordingly. It should store some type of tag or version information that it can use to verify that the data belongs to the codec. The codec should not dispose of the handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecScheduleFrame

Undocumented

```
ComponentResult ImageCodecScheduleFrame (
    ComponentInstance      ci,
    const ImageSubCodecDecompressRecord *drp,
```

```
ImageCodecTimeTriggerUPP      triggerProc,
void                          *triggerProcRefCon );
```

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

drp

A pointer to an `ImageSubCodecDecompressRecord` (IV–2289) structure.

triggerProc

An `ImageCodecTimeTriggerProc` (III–2096) callback.

triggerProcRefCon

A reference constant that is passed to your `ImageCodecTimeTriggerProc` callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCodec.h`

ImageCodecSetSettings

Sets the settings of an optional image codec dialog box.

```
ComponentResult ImageCodecSetSettings (
    ComponentInstance  ci,
    Handle             settings );
```

ci

An image compressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

settings

A handle to internal settings originally returned by either `ImageCodecRequestSettings` (II–735) or `ImageCodecGetSettings` (II–719). The codec should set its internal settings to match those of the settings handle. Because the codec does not own the handle, it should not dispose of it and should copy only its contents, not the handle itself. If the settings handle passed is empty, the codec should set its internal settings to a default state.

ImageCodecSetTimeBase

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function allows a codec to return its private settings. Set the codec’s internal settings to the state specified in the settings handle. The codec should always check the validity of the contents of the handle so that invalid settings are not used.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Base Image Decompressor Functions” (V–2805)

See Also

For the corresponding get function, see ImageCodecGetSettings (II–719).

ImageCodecSetTimeBase

Sets the time base for an image codec component.

```
ComponentResult ImageCodecSetTimeBase (  
    ComponentInstance  ci,  
    void               *base );
```

ci

An image codec component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

base

A pointer to the time base for this operation. Your application obtains this time base identifier from NewTimeBase (II–1119).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

ImageCodecSetTimeCode

Sets the timecode for the next frame that is to be decompressed.


```
ComponentResult ImageCodecSetTimeCode (
    ComponentInstance    ci,
    void                 *timeCodeFormat,
    void                 *timeCodeTime );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

timeCodeFormat

A pointer to a `TimeCodeDef` (IV–2482) structure. This structure contains the timecode definition information for the next frame to be decompressed.

timeCodeTime

A pointer to a `TimeCodeRecord` (IV–2484) structure. This structure contains the time value for the next frame in the current sequence.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your component receives this call whenever an application calls `SetDSequenceTimeCode` (III–1590). That function allows an application to set the timecode for a frame that is to be decompressed. The timecode information you receive applies to the next frame to be decompressed and is provided to the decompressor by `ImageCodecBandDecompress` (II–689).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecSourceChanged

Notifies your codec that one of the data sources has changed when an application calls `CDSequenceSetSourceData` (I–86) or `CDSequenceChangedSourceData` (I–76).

```
ComponentResult ImageCodecSourceChanged (
    ComponentInstance    ci,
    UInt32               majorSourceChangeSeed,
    UInt32               minorSourceChangeSeed,
    CDSequenceDataSourcePtr sourceData,
    long                 *flagsOut );
```

ImageCodecStandardParameterDialogDoAction

`ci`

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`majorSourceChangeSeed`

An integer value that is incremented each time a data source is added or removed. This provides an easy way for a codec to know when it needs to redetermine which data source inputs are available.

`minorSourceChangeSeed`

An integer value that is incremented each time a data source is added or removed, or the data contained in any of the data sources changes. This provides a way for a codec to know if the data available to it has changed.

`sourceData`

A pointer to a `CDSequenceDataSource` (IV–2165) structure. This structure contains a linked list of all data sources. Because each data source contains a link to the next data source, a codec can access all data sources from this structure.

`flagsOut`

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flagsOut Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

Programming summary: “Base Image Decompressor Functions” (V–2805)

ImageCodecStandardParameterDialogDoAction

Allows you to control the behavior of a standard parameter dialog box created by `ImageCodecCreateStandardParameterDialog` (II–692).

```
ComponentResult ImageCodecStandardParameterDialogDoAction (
    ComponentInstance ci,
    QTParameterDialog createdDialog,
    long action,
    void *params );
```

ci

An effect component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131). This must be the same instance as was passed to `ImageCodecCreateStandardParameterDialog` (II–692) to create the dialog box.

createdDialog

A reference to the dialog box created by the call to `ImageCodecCreateStandardParameterDialog`.

action

The action selector (see below), which determines which of the available actions you want the function to perform.

params

The (optional) parameter to the action. The type passed in this parameter depends on the value of the *action* parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

action Constants

`pdActionConfirmDialog`

Retrieves the current parameter values from the standard parameters dialog box. The parameter values are placed in the *parameters* parameter that was passed to the appropriate call to `ImageCodecCreateStandardParameterDialog` (II–692). Use this action when the user clicks the OK button in a dialog box your application has created that incorporates controls from the standard parameter dialog. If you created a stand-alone standard parameters dialog box, this action is automatically called for you when the user clicks the OK button.

`pdActionSetAppleMenu`

Use this selector to make the Apple menu available while the standard parameter dialog box is active. Pass the `MenuHandle` of the Apple menu in the *params* parameter.

`pdActionSetEditMenu`

Use this action to make your application’s Edit menu available while the standard parameters dialog box is active. Pass the `MenuHandle` of the Edit menu in the *params* parameter.

`pdActionGetDialogValues`

Retrieves the current parameter values from the standard parameters dialog box. The parameter values are placed in the QT atom container you pass in the *params* parameter to this function.

ImageCodecStandardParameterDialogDoAction

`pdActionSetPreviewUserItem`

Passes the number of the user item that should be replaced by the effect **preview** movie clip. The `params` parameter contains a long integer holding the user item number.

`pdActionSetPreviewPicture`

Use this action to change the images that are used in the **preview** window of the standard parameters dialog box. QuickTime provides default images but you may wish, for example, to use thumbnail images taken from your application instead. The `params` parameter contains a `QTParamPreviewPtr` that references the image to use.

`pdActionSetColorPickerEventProc`

Set the function that will be used when the standard parameters dialog box needs to display a color picker. The `params` parameter should contain a pointer to the replacement color picker function.

`pdActionSetDialogTitle`

Sets the title of the standard parameters dialog box. The `params` parameter should contain a Pascal `Str255` string.

`pdActionGetSubPanelMenu`

Returns a list of the sub-panels used by the standard parameters dialog box. If there are more parameter controls than can fit into the available area of the dialog box, the parameters are automatically split into sub-panels (usually by segmenting the set of parameters by group). The pop-up menu used to switch between the panels is returned as a `MenuHandle` in the `params` parameter.

`pdActionActivateSubPanel`

Sets the current sub-panel. The `params` parameter contains a long integer that is the number of the panel to be displayed.

`pdActionConductStopAlert`

Displays a Stop alert. The `params` parameter contains a `StringPtr` that is displayed in the alert. This feature can be used in conjunction with the `ImageCodecValidateParameters` (II-745) function to inform the user that invalid parameter values have been entered.

Discussion

This function allows you to change the default behavior of the standard parameter dialog box, and provides a mechanism for your application to communicate with controls that were incorporated into an application dialog box. It also allows you to retrieve parameter values from the dialog box at any time. You specify which function will be performed by passing an action selector in the `action` parameter and, optionally, a single parameter in the `params` parameter. Some of the actions you can specify through this function are only appropriate if you have incorporated

standard parameter dialog box controls within a dialog box created by your application.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

Programming summary: “Low-Level Effects Functions” (V–2898)

See Also

This function is only used with dialog boxes created by the low-level ImageCodecCreateStandardParameterDialog (II–692) function. For dialog boxes created by the high-level QTCreateStandardParameterDialog (II–1161) function, use QTStandardParameterDialogDoAction (II–1313).

ImageCodecTrimImage

Notifies your component whenever an application calls TrimImage (III–1956).

```
ComponentResult ImageCodecTrimImage (
    ComponentInstance      ci,
    ImageDescriptionHandle Desc,
    Ptr                    inData,
    long                   inBufferSize,
    ICMDataProcRecordPtr   dataProc,
    Ptr                    outData,
    long                   outBufferSize,
    ICMFlushProcRecordPtr  flushProc,
    Rect                   *trimRect,
    ICMProgressProcRecordPtr progressProc );
```

ci

An image decompressor component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

Desc

A handle to the ImageDescription (IV–2285) structure that describes the compressed image. Your component updates this structure to refer to the resized image.

inData

A pointer to the compressed image data. If the entire compressed image cannot be stored at this location, the application may provide a data-loading function;

ImageCodecTrimImage

see the description of the `dataProc` parameter to this function for details. This is a **32-bit clean** address.

`inBufferSize`

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If the application did not specify a data-loading function, this parameter is `NIL`.

`dataProc`

A pointer to an `ICMDataProcRecord` (IV–2279) structure. If the application did not provide a data-loading function, this parameter is `NIL`. In this case, the entire image must be in memory at the location specified by the `inData` parameter. If the data stream is not all in memory when the application calls `GetCompressedImageSize` (I–396), your component may call an application function that loads more compressed data.

`outData`

A pointer to a buffer to receive the trimmed image. If there is not sufficient memory to store the compressed image, the application may choose to write the compressed data to mass storage during the compression operation. The `flushProc` parameter identifies the data-unloading function. This is a **32-bit clean** address.

`outBufferSize`

The size of the buffer to be used by the data-unloading function specified by the `flushProc` parameter. If the application did not specify a data-unloading function, this parameter is `NIL`.

`flushProc`

A pointer to an `ICMFlushProcRecord` (IV–2280) structure. If the application did not provide a data-unloading function, this parameter is `NIL`. In this case, your component writes the entire compressed image into the memory location specified by the `outData` parameter. If there is not enough memory to store the compressed image, your component may call an application function that unloads some of the compressed data.

`trimRect`

A pointer to a `Rect` (IV–2399) structure that defines the desired image dimensions. Your component adjusts the structure's values so that they refer to the same rectangle in the resulting image. This is necessary whenever data is removed from the beginning of the image.

`progressProc`

A pointer to an `ICMProgressProcRecord` (IV–2284) structure. During the operation, your component should occasionally call an application function to

report its progress. If the application did not provide a **progress function**, this parameter is NIL.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

ImageCodecValidateParameters

Validates effect parameters.

```
ComponentResult ImageCodecValidateParameters (
    ComponentInstance      ci,
    QTAtomContainer        parameters,
    QTParameterValidationOptions  validationFlags,
    StringPtr              errorString );
```

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

parameters

The atom container containing the effect parameters to be validated.

validationFlags

Constants (see below) that control validation.

errorString

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

validationFlags Constants

`kParameterValidationNoFlags`

This value indicates that a standard validation should take place. This function is being called because the user has changed the value of a parameter in the standard parameters dialog box.

`kParameterValidationFinalValidation`

This value indicates that this validation is the final validation before the standard parameters dialog box is dismissed. This is useful if you want to perform a single validation of the parameters just after the user clicks the OK button to dismiss the dialog box.

ImageFieldSequenceBegin

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

ImageFieldSequenceBegin

Initiates an image field sequence operation and specifies the input and output data format.

```
OSErr ImageFieldSequenceBegin (
    ImageFieldSequence    *ifs,
    ImageDescriptionHandle desc1,
    ImageDescriptionHandle desc2,
    ImageDescriptionHandle descOut );
```

ifs

On return, contains the unique sequence identifier assigned to the sequence.

desc1

An ImageDescription (IV–2285) structure describing the format and characteristics of the data to be passed to ImageFieldSequenceExtractCombine (II–747) through the *data1* parameter.

desc2

An ImageDescription (IV–2285) structure describing the format and characteristics of the data to be passed to the ImageFieldSequenceExtractCombine function through the *data2* parameter. Set to NIL if the requested operation uses only one input frame.

descOut

The desired format of the resulting frames. Typically this is the same format specified by the *desc1* and *desc2* parameters.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Discussion

Use this function to set up an image field sequence operation and specify the input and output data format.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Video Fields” (V–2821)

ImageFieldSequenceEnd

Ends an image field sequence operation.

```
OSErr ImageFieldSequenceEnd (  
    ImageFieldSequence  ifs );
```

ifs

The unique sequence identifier that was returned by the `ImageFieldSequenceBegin` (II–746) function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You must call this function to terminate an image field sequence operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Video Fields” (V–2821)

ImageFieldSequenceExtractCombine

Performs field operations on video data.

```
OSErr ImageFieldSequenceExtractCombine (  
    ImageFieldSequence  ifs,  
    long                fieldFlags,  
    void                *data1,  
    long                dataSize1,  
    void                *data2,  
    long                dataSize2,  
    void                *outputData,  
    long                *outDataSize );
```

ifs

The unique sequence identifier that was returned by `ImageFieldSequenceBegin` (II–746).

ImageFieldSequenceExtractCombine

`fieldFlags`

Flags (see below) that specify the operation to be performed. A correctly formed request will specify two input fields, mapping one to the odd output field and the other to the even output field.

`data1`

A pointer to a buffer containing the data of input field one.

`dataSize1`

The size of the `data1` buffer.

`data2`

A pointer to a buffer containing the data of input field two. Set to `NIL` if the requested operation uses only one input frame.

`dataSize2`

The size of the `data2` buffer. Set to 0 if the requested operation uses only one input frame.

`outputData`

A pointer to a buffer to receive the resulting frame. Use `GetMaxCompressionSize` (I-420) to determine the amount of memory to allocate for this buffer.

`outDataSize`

On output this parameter returns the actual size of the data.

function result Returns the `codecUnimpErr` result code if there is no codec present in the system that can perform the requested operation. See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

fieldFlags Constants

`evenField1ToEvenFieldOut`

Maps the even field specified by the `data1` parameter to the even output field.

`evenField1ToOddFieldOut`

Maps the even field specified by the `data1` parameter to the odd output field.

`oddField1ToEvenFieldOut`

Maps the odd field specified by the `data1` parameter to the even output field.

`oddField1ToOddFieldOut`

Maps the odd field specified by the `data1` parameter to the odd output field.

`evenField2ToEvenFieldOut`

Maps the even field specified by the `data2` parameter to the even output field.

`evenField2ToOddFieldOut`

Maps the even field specified by the `data2` parameter to the odd output field.

`oddField2ToEvenFieldOut`

Maps the odd field specified by the `data2` parameter to the even output field.

`oddField2ToOddFieldOut`

Maps the odd field specified by the `data2` parameter to the odd output field.

Discussion

This function provides a method for working directly with fields of **interlaced** video. You can use it to change the field dominance of an image by reversing the two fields, or to create or remove the effects of the 3:2 pulldown commonly performed when transferring film to NTSC videotape. Because this function operates directly on the compressed video data, it is faster than working with decompressed images. It also has the added benefit of eliminating any image quality degradation that might result from lossy codecs.

This function accepts one or two compressed images as input and creates a single compressed image on output. You specify the operation to be performed using the `fieldFlags` parameter.

Special Considerations

The Apple Component Video (YUV) and Motion JPEG codecs currently support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Video Fields” (V-2821)

ImageTranscodeDisposeFrameData

Disposes transcoded image data.

```
OSErr ImageTranscodeDisposeFrameData (
    ImageTranscodeSequence  its,
    void                    *dstData );
```

`its`

The image transcoder sequence that was used to generate the transcoded data.

`dstData`

A pointer to the transcoded image data generated by the `ImageTranscodeFrame` (II-750) function.

ImageTranscodeFrame

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

When the transcoded image data returned by ImageTranscodeFrame (II–750) is no longer needed, use this function to dispose of the data. Only the image transcoder that generated the data can properly dispose of it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Image Transcoder Support” (V–2873)

ImageTranscodeFrame

Transcodes a frame of image data.

```
OSErr ImageTranscodeFrame (
    ImageTranscodeSequence  its,
    void                    *srcData,
    long                    srcDataSize,
    void                    **dstData,
    long                    *dstDataSize );
```

its

The image transcoder sequence to use to perform the transcoding operation.

srcData

A pointer to the source data to transcode.

srcDataSize

The size of the compressed source image data in bytes.

dstData

On return, a pointer to the transcoded image data.

dstDataSize

On return, the size of the transcoded image data.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

After creating the image transcoder sequence, using ImageTranscodeSequenceBegin (II–754), use this function to transcode a frame of image data. The caller is responsible for disposing of the transcoded data using ImageTranscodeDisposeFrameData (II–749).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Image Transcoder Support” (V-2873)

ImageTranscoderBeginSequence

Initiates an image transcoding sequence and specifies the input data format.

```
ComponentResult ImageTranscoderBeginSequence (
    ImageTranscoderComponent    itc,
    ImageDescriptionHandle       srcDesc,
    ImageDescriptionHandle       *dstDesc,
    void                         *data,
    long                         dataSize );
```

itc

The image transcoder component.

srcDesc

The ImageDescription (IV-2285) structure for the source compressed image data.

dstDesc

On return, a new ImageDescription structure.

data

First frame of data to be transcoded (may be NIL).

dataSize

Size of compressed image data pointed to by the data.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Discussion

This function specifies the format of source compressed image data in the *srcDesc* parameter. The image transcoder should allocate a new ImageDescription (IV-2285) structure and return it in the *dstDesc* parameter. The new ImageDescription structure should be a completely filled out image description which is sufficient for correctly decompressing the data generated by subsequent calls to ImageTranscoderConvert (II-752).

Version Notes

Introduced in QuickTime 3 or earlier.

ImageTranscoderConvert

Programming Info

C interface file: ImageCompression.h

Programming summary: “Image Transcoder Support” (V–2873)

ImageTranscoderConvert

Performs image transcoding operations.

```
ComponentResult ImageTranscoderConvert (  
    ImageTranscoderComponent    itc,  
    void                        *srcData,  
    long                        srcDataSize,  
    void                        **dstData,  
    long                        *dstDataSize );
```

itc

The image transcoder component.

srcData

A pointer to the source compressed image data to transcode.

srcDataSize

The size of the source image data, in bytes.

dstData

On return, a pointer to the transcoded data.

dstDataSize

On return, the size of the transcoded data in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The image transcoder component is responsible for allocating storage for the transcoded data, transcoding the data, and returning a pointer to the transcoded data in the `dstData` parameter. The size of the transcoded data in bytes should be returned in the `dstDataSize` parameter. The caller is responsible for disposing of the transcoded data using `ImageTranscoderDisposeData` (II–753).

The memory allocated to store the transcoded image data must not be in an unlocked handle. Even if the image transcoding operation can be performed in place, the transcoded data must be placed in a separate block of memory from the source data. The image transcoder component must not write back into the source image data.

The responsibility for allocating the buffer for the transcoded data has been placed in the transcoder with the intent that some hardware manufacturers may find it useful to place the transcoded data directly into on-board memory on their video board. If the transcoding operation is being performed on a QuickTime movie, the transcoded data pointer will be almost immediately passed on to a decompressor. If the decompressor is implemented in hardware, performance may be increased because the transcoded data is already loaded onto the decompression hardware.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Image Transcoder Support” (V-2873)

ImageTranscoderDisposeData

Disposes of transcoded data.

```
ComponentResult ImageTranscoderDisposeData (
    ImageTranscoderComponent    itc,
    void                        *dstData );
```

itc

The image transcoder component.

dstData

A pointer to the transcoded data.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

When the client of the image transcoder component is done with a piece of transcoded data, this function must be called with a pointer to the transcoded data. The image transcoder component should not make any assumptions about the maximum number of outstanding pieces of transcoded data or the order in which the transcoding data will be disposed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Image Transcoder Support” (V-2873)

ImageTranscoderEndSequence

ImageTranscoderEndSequence

Ends an image transcoding sequence.

```
ComponentResult ImageTranscoderEndSequence (  
    ImageTranscoderComponent    itc );
```

itc

The image transcoder component whose transcoder sequence is ending.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

`ImageTranscoderEndSequence` is called when there are no more frames of data to be transcoded using the parameters specified in the previous call to `ImageTranscoderBeginSequence` (II–751). After calling this function, the component will either be closed or receive another call to `ImageTranscoderBeginSequence` with a different `ImageDescription` (IV–2285) structure. For example, the dimensions of the source image may be different.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Image Transcoder Support” (V–2873)

ImageTranscodeSequenceBegin

Initiates an image transcoder sequence operation.

```
OSErr ImageTranscodeSequenceBegin (  
    ImageTranscodeSequence    *its,  
    ImageDescriptionHandle     srcDesc,  
    OSType                    destType,  
    ImageDescriptionHandle     *dstDesc,  
    void                      *data,  
    long                      dataSize );
```

its

The image transcoder sequence identifier. If the operation fails, the value pointed to is set to `NIL`.

srcDesc

The ImageDescription (IV–2285) structure for the source compressed image data.

destType

The desired compression format into which to transcode the source data.

dstDesc

On return, an ImageDescription (IV–2285) structure for the data which will be generated by the image transcoding sequence.

data

A pointer to first frame of compressed data to transcode. Set to NIL if not available.

dataSize

The size of the compressed data, in bytes. Set to 0 if no data is provided.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error. If no transcoder is available to perform the requested transcoding operation, a cantFindHandler error is returned.

Discussion

This function begins an image transcoder sequence operation and returns the sequence identifier in the `its` parameter. The caller is responsible for disposing of the ImageDescription structure that is returned in the `dstDesc` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Image Transcoder Support” (V–2873)

ImageTranscodeSequenceEnd

Ends an image transcoder sequence operation.

```
OSErr ImageTranscodeSequenceEnd (
    ImageTranscodeSequence  its );
```

`its`

The identifier of the image transcoder sequence to dispose. It is safe to pass a value of 0 in this parameter.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

InitializeQHdr

Discussion

You must call this function to terminate an image transcoder sequence operation and dispose of the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Image Transcoder Support” (V–2873)

InitializeQHdr

Initializes a Windows QHdr (IV–2345) structure for use by QuickTime.

```
void InitializeQHdr (  
    QHdr    *qhdr );
```

qhdr

A pointer to a QHdr (IV–2345) structure.

Discussion

The `InitializeQHdr` function initializes the various fields of the Windows queue header to startup values and associates a mutex with the queue to provide safe access via `Enqueue` (I–336) and `Dequeue` (I–253). The mutex identifier is stored in the `MutexID` field of the QHdr (IV–2345) structure. Your application or component is not required to manage this mutex; `Enqueue` and `Dequeue` will handle this for you. A QHdr structure is typically used by QuickTime image decompressor components to manage frame queues. Once you are done with the queue, call `TerminateQHdr` (III–1926) to free the mutex.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QTML.h`

InitializeQTML

Initializes the QuickTime Media Layer.

```
OSErr InitializeQTML (  
    long    flag );
```

flag

Flags (see below) that specify several options.

function result Returns an error if QuickTime is not installed or cannot be initialized. See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flag Constants

`kInitializeQTMLNoSoundFlag`

If this flag is set, the Sound Manager is not initialized and therefore no sound APIs will be supported during the session. Use this flag only if no sound support is needed.

`kInitializeQTMLUseGDIFlag`

If this flag is set, all drawing will be done using graphics device interface (GDI) calls; neither `DirectDraw` nor DCI services will be used for onscreen graphics support. If this flag is not set, QuickTime will try to use `DirectDraw`, and then DCI, to support direct-to-surface graphics support and take advantage of any hardware acceleration provided by these services, falling back to GDI only if `DirectDraw` and DCI are unavailable. You should normally not set this flag.

`kInitializeQTMLDisableDirectSound`

Disable QTML’s use of `DirectSound`.

`kInitializeQTMLUseExclusiveFullScreenModeFlag`

Operate exclusively in full screen mode, in versions of QuickTime later than 3.0.

Discussion

Use this function to initialize a QTML session before calling `EnterMovies` (I–337). If you are writing a routine that does not know from context whether this function has already been called, add a call to it at the beginning of the routine and a call to `TerminateQTML` (III–1926) at the end. It does no harm to call this function more than once, as long as each call is nested with a matching call to `TerminateQTML`. If this function has already been called, subsequent calls do nothing except increment a counter. Calls to `TerminateQTML` just decrement the counter (if it is nonzero). Only the first nested call to this function and the last nested call to `TerminateQTML` do any actual work, so there is no penalty for having nested calls.

Special Considerations

You should not make this call from a QuickTime component such as an image decompressor; it is provided only for host applications.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QTML.h`

InitializeQTS

Related Java Methods

`quicktime.QTSession.initialize()`, `quicktime.QTSession.open()`

InitializeQTS

Initializes QuickTime streaming.

`OSErr InitializeQTS ( void );`

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeStreaming.h`

InitializeQTVR

Initializes QuickTime virtual reality.

`OSErr InitializeQTVR ( void );`

function result See “Error Codes” (IV–2663). If this function is unable to load the `QuickTimeVR.qtx` file, it returns error code `qtvrlibraryLoadErr`. If you call any other function of QuickTime VR before calling this function or after this function has failed to load the `QuickTimeVR.qtx` file, it returns error code `qtvruninitialized`. It returns `noErr` if there is no error.

Discussion

This function and `TerminateQTVR` (III–1927) are required for QuickTime VR to run in a Windows environment. They do nothing in the Mac OS environment, but must be included for cross-platform compatibility. When your application calls this function in the Windows environment, the code attempts to find the `QuickTimeVR.qtx` file through the normal search paths.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Initializing and Terminating QuickTime VR” (V–2904)

Related Java Methods

```
quicktime.QTSession.initializeVR()
```

InsertEmptyMovieSegment

Adds an empty segment to a movie.

```
OSErr InsertEmptyMovieSegment (
    Movie      dstMovie,
    TimeValue   dstIn,
    TimeValue   dstDuration );
```

dstMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), or `NewMovieFromHandle` (II-1084).

dstIn

A time value that specifies where the segment is to be inserted. This time value must be expressed in the movie's time scale.

dstDuration

A time value that specifies the duration of the segment to be added. This time value must be expressed in the movie's time scale.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

You specify the starting time and duration of the empty segment to be added. These times must be expressed in the movie's time scale. You cannot add empty space to the end of a movie. If you want to insert a segment beyond the end of a movie, use `InsertMovieSegment` (II-762).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Low-Level Movie Editing Functions” (V-2749)

Related Java Methods

```
quicktime.std.movies.Movie.insertEmptySegment()
```

InsertEmptyTrackSegment

InsertEmptyTrackSegment

Adds an empty segment to a track.

```
OSErr InsertEmptyTrackSegment (
    Track          dstTrack,
    TimeValue      dstIn,
    TimeValue      dstDuration );
```

dstTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

dstIn

A time value specifying where the segment is to be inserted. This time value must be expressed in the time scale of the movie that contains the destination track.

dstDuration

A time value that specifies the duration of the segment to be added. This time value must be expressed in the time scale of the movie that contains the destination track.

function result See “Error Codes” (IV–2663). If you try to add an empty segment beyond the end of a track, this function does not add the empty segment and returns a result code of `invalidTime`. Returns `noErr` if there is no error.

Discussion

You specify the starting time and duration of the empty segment to be added. These times must be expressed in the movie’s time scale. This function then inserts the appropriate amount of empty time into the track. The exact meaning of the term empty time depends upon the type of track. For example, empty time in a sound track is silence. Note that you cannot add empty space to the end of a movie or to the end of a track.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Tracks” (V–2750)

Related Java Methods

`quicktime.std.movies.Track.insertEmptySegment()`

InsertMediaIntoTrack

Inserts a reference to a media segment into a track.

```
OSErr InsertMediaIntoTrack (
    Track          theTrack,
    TimeValue      trackStart,
    TimeValue      mediaTime,
    TimeValue      mediaDuration,
    Fixed          mediaRate );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) or `GetMovieTrack` (I–497).

trackStart

A time value specifying where the segment is to be inserted. This time value must be expressed in the movie’s time scale. If you set this parameter to –1, the media data is added to the end of the track.

mediaTime

A time value specifying the starting point of the segment in the media. This time value must be expressed in the media’s time scale.

mediaDuration

A time value specifying the duration of the media’s segment. This time value must be expressed in the media’s time scale.

mediaRate

The media’s rate. A value of 1.0 indicates the media’s natural playback rate. This value should be positive and not 0.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You specify the segment in the media by providing a starting time and duration. You specify the point in the destination track by providing a time in the track. `InsertMediaIntoTrack` then inserts the media segment into the track at the specified location. The Movie Toolbox determines the duration of the segment in the track based on the media rate and duration information you provide.

Use this function after you have added samples to a media. If you play the track before you call this function, the track does not contain the new media data.

InsertMovieSegment

Here's an example of using this function to add atom containers to a track:

```
//InsertMediaIntoTrack coding example
long descSize;
QTVRSampleDescriptionHandle qtvrSampleDesc;

// Create a QTVR sample description handle
descSize = sizeof(QTVRSampleDescription) + GetHandleSize((Handle) vrWorld)
           - sizeof(UInt32);
qtvrSampleDesc = (QTVRSampleDescriptionHandle) NewHandleClear (descSize);
(*qtvrSampleDesc)->size = descSize;
(*qtvrSampleDesc)->type = kQTVRQTVRType;

// Copy the VR world atom container data into the QTVR sample description
BlockMove (*((Handle) vrWorld), &((*qtvrSampleDesc)->data),
           GetHandleSize((Handle) vrWorld));
// Now add it to the QTVR track's media
err = BeginMediaEdits (qtvrMedia);
err = AddMediaSample (qtvrMedia, (Handle) nodeInfo, 0,
                     GetHandleSize((Handle) nodeInfo), duration,
                     (SampleDescriptionHandle) qtvrSampleDesc, 1, 0, &sampleTime);
err = EndMediaEdits (qtvrMedia);
InsertMediaIntoTrack (qtvrTrack, trackTime, sampleTime, duration, 1L<<16);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Editing Tracks” (V-2750)

Related Java Methods

quicktime.std.movies.Track.insertMedia()

InsertMovieSegment

Copies part of one movie to another.

```
OSErr InsertMovieSegment (
    Movie      srcMovie,
    Movie      dstMovie,
    TimeValue  srcIn,
```



```

    TimeValue    srcDuration,
    TimeValue    dstIn );

```

srcMovie

The source movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084). This function obtains the movie segment from the source movie specified in this parameter.

dstMovie

The destination movie for this operation. The function places a copy of the segment, which it obtained from the source movie, into this destination movie.

srcIn

The start of the segment in the source movie. This time value must be expressed in the source movie’s time scale.

srcDuration

The duration of the segment in the source movie. This time value must be expressed in the source movie’s time scale.

dstIn

A time value specifying where the segment is to be inserted. This time value must be expressed in the destination movie’s time scale.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

If you are not copying data from one location in a movie to a different point in the same movie, this function may create new tracks, as appropriate. Before adding a track to the destination movie, the Movie Toolbox looks in the destination movie for tracks that have the same characteristics as the tracks in the source movie. The toolbox considers several characteristics when searching for an appropriate track, including track spatial dimensions, track matrix, track clipping region, track matte, alternate group affiliation, media time scale, media type, media language, and data reference (that is, referring to the same file). If the Movie Toolbox cannot find an appropriate track in the destination movie, it creates a new track with the proper characteristics.

Special Considerations

If you have assigned a **progress function** to the destination movie, the Movie Toolbox calls that progress function during long copy operations. Some Movie Toolbox functions can take a long time to execute. For example, if you call `FlattenMovie` (I–372) and specify a large movie, the Movie Toolbox must read and

InsertTrackSegment

write all the sample data for the movie. During such operations you may wish to display some kind of progress indicator to the user.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Low-Level Movie Editing Functions” (V-2749)

Related Java Methods

`quicktime.std.movies.Movie.insertSegment()`

InsertTrackSegment

Copies data into a track.

```
OSErr InsertTrackSegment (
    Track          srcTrack,
    Track          dstTrack,
    TimeValue      srcIn,
    TimeValue      srcDuration,
    TimeValue      dstIn );
```

`srcTrack`

The source track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II-1092) and `GetMovieTrack` (I-497).

`dstTrack`

The destination track for this operation. This function places a copy of the segment, which is obtained from the source track, into this destination track. The media for the destination track must be opened for writing by calling `BeginMediaEdits` (I-56) in order for the data to be copied. If the media is not opened for writing, the segment will be copied by reference. At the end of the editing session, your application must call `EndMediaEdits` (I-335) if it has called `BeginMediaEdits`.

`srcIn`

The start of the segment in the source track. This time value must be expressed in the time scale of the movie that contains the source track.

`srcDuration`

The duration of the segment in the source track. This time value must be expressed in the time scale of the movie that contains the source track.

`dstIn`

A time value specifying where the segment is to be inserted. This time value must be expressed in the time scale of the movie that contains the destination track.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

If you are copying data between tracks, make sure that the two tracks are of the same type. For example, you cannot copy a segment from a sound track into a video track. If you have assigned a **progress function** to the movie that contains the destination track, the Movie Toolbox calls that progress function during long copy operations.

Special Considerations

If you copy a segment without calling `BeginMediaEdits` on the destination track’s media, the data can be copied later by flattening the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Tracks” (V-2750)

Related Java Methods

`quicktime.std.movies.Track.insertSegment()`

InstrumentCloseComponentResFile

Closes a **resource** file.

```
ComponentResult InstrumentCloseComponentResFile (
    ComponentInstance ci,
    short resFile );
```

`ci`

An instrument component instance. Your software obtains this reference from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131).

`resFile`

A reference to a **resource** file, returned previously by `InstrumentOpenComponentResFile` (II-770).

InstrumentGetComponentRefCon

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Instrument Component Functions” (V–2923)

InstrumentGetComponentRefCon

Obtains the reference constant for an instrument component.

```
ComponentResult InstrumentGetComponentRefCon (  
    ComponentInstance    ci,  
    void                 **refCon );
```

ci

An instrument component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

refCon

A reference constant.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Instrument Component Functions” (V–2923)

See Also

For the corresponding set function, see InstrumentSetComponentRefCon (II–771).

InstrumentGetInfo

Returns information about all the atomic instruments supported by an instrument component.

```
ComponentResult InstrumentGetInfo (  
    ComponentInstance    ci,  
    long                 getInstrumentInfoFlags,  
    InstCompInfoHandle   *instInfo );
```

ci

An instrument component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

getInstrumentInfoFlags

Flags (see below) that specify whether you want a list of fixed instruments, modifiable instruments, or all instruments.

instInfo

On return, an `InstCompInfo` (IV–2291) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

getInstrumentInfoFlags Constants

`kGetInstrumentInfoNoBuiltIn`

Don’t return built-in instruments.

`kGetInstrumentInfoMidiUserInst`

Do return user instruments for a MIDI device.

`kGetInstrumentInfoNoIText`

Don’t return international text strings.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Instrument Component Functions” (V–2923)

InstrumentGetInst

Returns an atomic instrument.

```
ComponentResult InstrumentGetInst (
    ComponentInstance  ci,
    long               instID,
    AtomicInstrument    *atomicInst,
    long               flags );
```

ci

An instrument component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

instID

The instrument component instrument ID from the `InstCompInfo` (IV–2291) structure returned by `InstrumentGetInfo` (II–766).

InstrumentGetSynthesizerType

`atomicInst`

On return, the atomic instrument.

`flags`

Constants (see below) that specify what pieces of information about an atomic instrument the caller is interested in.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`kGetAtomicInstNoExpandedSamples`

Eliminate the expanded samples.

`kGetAtomicInstNoOriginalSamples`

Eliminate the original samples.

`kGetAtomicInstNoSamples`

Eliminate both the original and expanded samples.

`kGetAtomicInstNoKnobList`

Eliminate the knob list.

`kGetAtomicInstNoInstrumentInfo`

Eliminate the About box information.

`kGetAtomicInstOriginalKnobList`

Include the original knob list.

`kGetAtomicInstAllKnobs`

Include the current knob list.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Instrument Component Functions” (V–2923)

InstrumentGetSynthesizerType

Obtains the synthesizer type for an instrument.

```
ComponentResult InstrumentGetSynthesizerType (
    ComponentInstance ci,
    OSType             *synthesizerType );
```

ci

An instrument component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

synthesizerType

Pointer to the returned type (see below).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

synthesizerType Constants

`kSoftSynthComponentSubType`

The QuickTime music synthesizer. This is the built-in synthesizer. Value is 'ss ' (3rd and 4th characters are spaces).

`kGMSynthComponentSubType`

The General MIDI synthesizer. Value is 'gm ' (3rd and 4th characters are spaces).

Discussion

This function returns the synthesizer type (the subtype of the synthesizer component) associated with a given instrument component.

Version Notes

Introduced in QuickTime 3 or earlier. Older instrument components (developed before QuickTime 2) may not support this call.

Programming Info

C interface file: `QuickTimeMusic.h`

InstrumentInitialize

Tells the base class instrument component how to interpret the instrument component **resources**.

```
ComponentResult InstrumentInitialize (
    ComponentInstance  ci,
    long               initFormat,
    void               *initParams );
```

ci

An instrument component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

initFormat

Set to 0.

InstrumentOpenComponentResFile

`initParams`

Set to NIL.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Used by developers of instrument components, this is a call the instrument component makes to the base class instrument component to tell it how to interpret the instrument component **resources**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Instrument Component Functions” (V–2923)

InstrumentOpenComponentResFile

Opens a **resource** file containing instruments in the instrument component and makes it the current resource file.

```
ComponentResult InstrumentOpenComponentResFile (  
    ComponentInstance    ci,  
    short                *resFile );
```

`ci`

An instrument component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`resFile`

On return, a **resource** reference.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Instrument Component Functions” (V–2923)

InstrumentSetComponentRefCon

Overrides SetComponentRefcon (III–1573) and sets an instrument component’s reference constant to a specified value.

```
ComponentResult InstrumentSetComponentRefCon (
    ComponentInstance    ci,
    void                *refCon );
```

ci

An instrument component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

refCon

A reference constant.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

See Also

For the corresponding get function, see InstrumentGetComponentRefCon (II–766).

InvalidateMovieRegion

Invalidates a small area of a movie.

```
OSErr InvalidateMovieRegion (
    Movie        theMovie,
    RgnHandle    invalidRgn );
```

theMovie

The movie whose area you wish to invalidate. Your application obtains this movie identifier from such functions as NewMovie (II–1069), NewMovieFromFile (II–1080), and NewMovieFromHandle (II–1084).

invalidRgn

A region indicating the area of the movie to invalidate. If necessary, QuickTime will make a copy of this region. To invalidate the entire movie area, pass NIL for this parameter.

InvalidateSprite

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Use this function instead of `UpdateMovie` (III–1981) to invalidate a small area of a movie. It marks all areas of the movie that intersect the `invalidRgn` parameter. The next time you call `MoviesTask` (II–973), the Movie Toolbox redraws the marked areas. This provides a way to invalidate a portion of the movie’s area instead of its entire area, as does `UpdateMovie`. This allows for higher performance update handling when a movie has many tracks or covers a large area. For handling of update events, applications should continue to use `UpdateMovie`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movies and Your Event Loop” (V–2720)

Related Java Methods

`quicktime.std.movies.Movie.invalidateRegion()`

InvalidateSprite

Invalidates the portion of a **sprite**’s sprite world that is occupied by a sprite.

```
void InvalidateSprite (
    Sprite    theSprite );
```

`theSprite`

The **sprite** for this operation.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

In most cases, you do not need to call this function. When you call `SetSpriteProperty` (III–1641) to modify a **sprite**’s properties, it takes care of invalidating the appropriate regions of the sprite world. However, you might call this function if you change a sprite’s image data but retain the same image data pointer.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Manipulating Sprites” (V–2901)

Related Java Methods

`quicktime.std.anim.Sprite.invalidate()`

InvalidateSpriteWorld

Invalidates a rectangular area of a **sprite** world.

```
OSErr InvalidateSpriteWorld (
    SpriteWorld    theSpriteWorld,
    Rect           *invalidArea );
```

`theSpriteWorld`

The **sprite** world for this operation.

`invalidArea`

A pointer to the `Rect` (IV–2399) structure that defines the area that should be invalidated. This rectangle should be specified in the **sprite** world’s source space, which is the coordinate system of the sprite layer’s graphics world before the sprite world’s matrix is applied to it. To invalidate the entire sprite world, pass `NIL` for this parameter.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Typically, your application calls this function when the **sprite** world’s destination window receives an update event. Invalidating an area of the sprite world will cause the area to be redrawn the next time that `SpriteWorldIdle` (III–1908) is called.

Special Considerations

When you modify **sprite** properties, invalidation takes place automatically; you do not need to call this function.

Version Notes

Introduced in QuickTime 3 or earlier.

InverseMatrix

Programming Info

C interface file: `Movies.h`

Programming summary: “Working with Sprite Worlds” (V–2900)

Related Java Methods

`quicktime.std.anim.SpriteWorld.invalidate()`

InverseMatrix

Creates a new matrix that is the inverse of a specified matrix.

```
Boolean InverseMatrix (
    const MatrixRecord    *m,
    MatrixRecord          *im );
```

m

A pointer to the source `MatrixRecord` (IV–2304) structure for the operation.

im

A pointer to a `MatrixRecord` (IV–2304) structure that is to receive the new matrix. The function updates this structure so that it contains a matrix that is the inverse of that specified by the *m* parameter.

function result A Boolean value of TRUE if `InverseMatrix` was able to create an inverse matrix, FALSE otherwise.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Matrices” (V–2764)

Related Java Methods

`quicktime.std.image.Matrix.inverse()`

IsMovieDone

Determines if a particular movie has completely finished playing.

```
Boolean IsMovieDone (
    Movie    theMovie );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result Returns TRUE if the specified movie has finished playing, otherwise returns FALSE.

Discussion

A movie with a positive rate (playing forward) is considered done when its movie time reaches the movie end time. Conversely, a movie with a negative rate (playing backward) is considered done when its movie time reaches the movie start time. If your application has changed the movie's active segment, the status returned by this function is relative to the active segment, rather than to the entire movie. You can use `SetMovieActiveSegment` (III–1609) to change a movie's active segment.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movies and Your Event Loop” (V–2720)

Related Java Methods

`quicktime.std.movies.Movie.isDone()`

IsScrapMovie

Checks the system scrap to find out if it can translate any of the data into a movie.

```
Component IsScrapMovie (
    Track    targetTrack );
```

`targetTrack`

The location of the potential target movie track for the data on the system scrap.

function result If `IsScrapMovie` finds an appropriate type, it returns a movie import component that can translate the scrap. Otherwise, it returns 0.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Movie Editing Functions” (V–2746)

IsTaskBarVisible

Related Java Methods

```
quicktime.std.movies.Track.isScrapMovie(),  
quicktime.std.qtcomponents.MovieImporter.fromTrack()
```

IsTaskBarVisible

Returns the current visibility state of the task bar.

```
Boolean IsTaskBarVisible ( void );
```

function result Returns TRUE if the task bar is visible, FALSE otherwise.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

ITextAddString

Undocumented

```
OSErr ITextAddString (
    QTAAtomContainer    container,
    QTAAtom             parentAtom,
    RegionCode          theRegionCode,
    ConstStr255Param    theString );
```

container

Undocumented

parentAtom

Undocumented

theRegionCode

A 16-bit signed integer containing an international **region code**; see “Localization Codes” (IV-2673).

theString

The string to add.

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.AtomContainer.iTextAddString()`

ITextGetString

Undocumented

```
OSErr ITextGetString (
    QTAtomContainer    container,
    QTAtom             parentAtom,
    RegionCode          requestedRegion,
    RegionCode          *foundRegion,
    StringPtr           theString );
```

`container`

Undocumented

`parentAtom`

Undocumented

`requestedRegion`

Undocumented

`foundRegion`

On return, a 16-bit signed integer containing an international **region code**; see “Localization Codes” (IV–2673).

`theString`

The found string.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.AtomContainer.iTextGetString()`

ITextRemoveString

ITextRemoveString

Undocumented

```
OSErr ITextRemoveString (
    QTAtomContainer    container,
    QTAtom             parentAtom,
    RegionCode         theRegionCode,
    long               flags );
```

container

Undocumented

parentAtom

Undocumented

theRegionCode

A 16-bit signed integer containing an international **region code**; see “Localization Codes” (IV–2673).

flags

Flags (see below) that modify the process.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

flags Constants

kITextRemoveEverythingBut

Undocumented

kITextRemoveLeaveSuggestedAlternate

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Related Java Methods

quicktime.std.movies.AtomContainer.iTextRemoveString()

L

L

LoadMediaIntoRam

Loads a media's data into memory.

```
OSErr LoadMediaIntoRam (
    Media      theMedia,
    TimeValue  time,
    TimeValue  duration,
    long       flags );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II-1120) and `GetTrackMedia` (I-535).

time

The starting time of the media segment to load. This time value must be expressed in the media's time coordinate system.

duration

The length of the segment to load. Use `GetMediaDuration` (I-430) to determine the length of the entire media. Note that the media handler may load more data than you specify if the media data was added in larger pieces.

flags

Flags (see below) that give you explicit control over what is loaded into memory and how long to keep it around. You can set these flags in any combination.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See "Error Codes" (IV-2663).

Flags Constants

keepInRam

Renders all data loaded with this flag set as nonpurgeable. Nonpurgeable data is not released from memory until you request it explicitly. This practice can fill up the user's heap very quickly; exercise caution.

unkeepInRam

Renders all indicated data purgeable. The data is not necessarily released from memory immediately, however. Information about whether a chunk can be purged is maintained internally by a single bit. This means there is no counter.

LoadMediaIntoRam

Therefore, if you care very much about the data, you have to work very hard and use the edit list meticulously.

`flushFromRam`

Purges all indicated data from memory, unless it is currently in use by a media handler (for example, if it is still drawing frames from the requested times). This flag makes the memory available for purging, and then performs the purge. You may want to use this option if you are particularly low on memory.

`loadForwardTrackEdits`

In some cases, an edited movie plays back much more smoothly if the data around edits is already in RAM. By setting either this flag or the `loadBackwardTrackEdits` flag, you can load only the data around edits. The Movie Toolbox walks through the edits and decides the right amount of data to load for you. If you are going to play the movie forward, set only the `loadForwardTrackEdits` flag. If you are going to play in both directions, or you don't know which direction, set both flags.

`loadBackwardTrackEdits`

In some cases, an edited movie plays back much more smoothly if the data around edits is already in RAM. By setting either this flag or `loadForwardTrackEdits`, you can load only the data around edits. The Movie Toolbox walks through the edits and decides the right amount of data to load for you. If you are going to play the movie only backward, set the `loadBackwardTrackEdits` flag. If you are going to play in both directions, or you don't know which direction, set both flags.

Discussion

The exact behavior of `LoadMediaIntoRam` is dependent on the media handler.

Special Considerations

If `LoadMediaIntoRam` fails because it is out of memory, no data is purged.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V-2722)

Related Java Methods

`quicktime.std.movies.media.Media.loadIntoRam()`

LoadMovieIntoRam

Loads a movie's data into memory.

```
OSErr LoadMovieIntoRam (
    Movie      theMovie,
    TimeValue  time,
    TimeValue  duration,
    long       flags );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), and `NewMovieFromHandle` (II-1084).

time

The starting time of the movie segment to load.

duration

The length of the segment to load. Use `GetMovieDuration` (I-470) to determine the length of the entire movie. Note that the Movie Toolbox may load more data than you specify due to the way the data is loaded.

flags

Flags (see below) that give you explicit control over what is loaded into memory and how long to keep it around. You can set these flags in any combination.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Flags Constants

keepInRam

Renders all data loaded with this flag set as nonpurgeable. Nonpurgeable data is not released from memory until you request it explicitly. This practice can fill up the user's heap very quickly; exercise caution.

unkeepInRam

Renders all indicated data purgeable. The data is not necessarily released from memory immediately, however. Information about whether a chunk can be purged is maintained internally by a single bit. This means there is no counter. Therefore, if you care very much about the data, you have to work very hard and use the edit list meticulously.

LoadTrackIntoRam

flushFromRam

Purges all indicated data from memory, unless it is currently in use by a media handler (for example, if it is still drawing frames from the requested times). This flag makes the memory available for purging, and then performs the purge. You may want to use this option if you are particularly low on memory.

loadForwardTrackEdits

In some cases, an edited movie plays back much more smoothly if the data around edits is already in RAM. By setting either this flag or the `loadBackwardTrackEdits` flag, you can load only the data around edits. The Movie Toolbox walks through the edits and decides the right amount of data to load for you. If you are going to play the movie forward, set only the `loadForwardTrackEdits` flag. If you are going to play in both directions, or you don't know which direction, set both flags.

loadBackwardTrackEdits

In some cases, an edited movie plays back much more smoothly if the data around edits is already in RAM. By setting either this flag or `loadForwardTrackEdits`, you can load only the data around edits. The Movie Toolbox walks through the edits and decides the right amount of data to load for you. If you are going to play the movie only backward, set the `loadBackwardTrackEdits` flag. If you are going to play in both directions, or you don't know which direction, set both flags.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V-2722)

Related Java Methods

`quicktime.std.movies.Movie.loadIntoRam()`

LoadTrackIntoRam

Loads a track's data into memory.

```
OSErr LoadTrackIntoRam (
    Track          theTrack,
    TimeValue      time,
    TimeValue      duration,
    long           flags );
```

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II-1092) and `GetMovieTrack` (I-497).

`time`

The starting time of the track segment to load. You must specify this time value in the movie's time coordinate system.

`duration`

The length of the segment to load. Use `GetTrackDuration` (I-529) to determine the length of the entire movie. Note that the media handler may load more data than you specify.

`flags`

Flags (see below) that give you explicit control over what is loaded into memory and how long to keep it around. You can set these flags in any combination.

function result If the track does not fit, the function returns an error. See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Flags Constants

`keepInRam`

Renders all data loaded with this flag set as nonpurgeable. Nonpurgeable data is not released from memory until you request it explicitly. This practice can fill up the user's heap very quickly; exercise caution.

`unkeepInRam`

Renders all indicated data purgeable. The data is not necessarily released from memory immediately, however. Information about whether a chunk can be purged is maintained internally by a single bit. This means there is no counter. Therefore, if you care very much about the data, you have to work very hard and use the edit list meticulously.

`flushFromRam`

Purges all indicated data from memory, unless it is currently in use by a media handler (for example, if it is still drawing frames from the requested times). This flag makes the memory available for purging, and then performs the purge. You may want to use this option if you are particularly low on memory.

`loadForwardTrackEdits`

In some cases, an edited movie plays back much more smoothly if the data around edits is already in RAM. By setting either this flag or the `loadBackwardTrackEdits` flag, you can load only the data around edits. The Movie Toolbox walks through the edits and decides the right amount of data to load for you. If you are going to play the movie forward, set only the

MACEVersion

`loadForwardTrackEdits` flag. If you are going to play in both directions, or you don't know which direction, set both flags.

`loadBackwardTrackEdits`

In some cases, an edited movie plays back much more smoothly if the data around edits is already in RAM. By setting either this flag or `loadForwardTrackEdits`, you can load only the data around edits. The Movie Toolbox walks through the edits and decides the right amount of data to load for you. If you are going to play the movie only backward, set the `loadBackwardTrackEdits` flag. If you are going to play in both directions, or you don't know which direction, set both flags.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Enhancing Movie Playback Performance" (V-2722)

Related Java Methods

`quicktime.std.movies.Track.loadIntoRam()`

M

MACEVersion

Obsolete.

`NumVersion MACEVersion ( void );`

Version Notes

Introduced in QuickTime 3 or earlier. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: `Sound.h`

MakeFilePreview

Creates a **preview** for a file.

```
OSErr MakeFilePreview (
    short          resRefNum,
    ICMProgressProcRecordPtr progress );
```

resRefNum

The **resource** file for this operation. You must have opened this resource file with write permission. If there is a **preview** in the specified file, the Movie Toolbox replaces that preview with a new one.

progress

A pointer to an ICMProgressProcRecord (IV–2284) structure. During the process of creating the **preview**, the Movie Toolbox may occasionally call a function you provide in order to report its progress. You can then use this information to keep the user informed.

Set this parameter to –1 to use the default **progress function**. If you specify a progress function, it must comply with the interface defined for Image Compression Manager progress functions; see “Image Compression Manager” in *Inside Macintosh: QuickTime* (listed in the **bibliography**) for more information. Set this parameter to NIL to prevent the Movie Toolbox from calling a progress function.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

You should create a **preview** whenever you save a movie. You specify the file by supplying a reference to its **resource** file. You must have opened this resource file with write permission. If there is a preview in the specified file, the Movie Toolbox replaces that preview with a new one.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Creating File Previews” (V–2757)

MakeImageDescriptionForEffect

Returns an ImageDescription (IV–2285) structure you can use to help create a sample description for an effect.

```
OSErr MakeImageDescriptionForEffect (
    OSType          effectType,
```

MakeImageDescriptionForEffect

```
ImageDescriptionHandle    *idh );
```

effectType

The four-character code identifying the type of effect to make an image description for. See “Effects Codes” (IV–2662).

idh

The handle of an `ImageDescription` (IV–2285) structure. On entry, this parameter normally points to an `ImageDescription` structure whose contents are NIL. On return, the structure is correctly filled out for the selected effect type.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

To create a sample description, you create and fill out a data structure of type `ImageDescription` (IV–2285). This function simplifies this process. Only sample descriptions made with this function can be used in stacked effects, where one effect track acts as a source for another.

The following sample code creates a sample description:.

```
// MakeImageDescriptionForEffect coding example
// Return a new image description with default and specified values.

ImageDescriptionHandle QTEffSeg_MakeSampleDescription (
                                OSType theEffectType, short theWidth, short theHeight)
{
    ImageDescriptionHandle    mySampleDesc = NIL;

#ifdef USES_MAKE_IMAGE_DESC_FOR_EFFECT
    OSErr                    myErr = noErr;

    // create a new sample description
    myErr = MakeImageDescriptionForEffect(theEffectType, &mySampleDesc);
    if (myErr != noErr)
        return(NIL);
#else
    // create a new sample description
    mySampleDesc = (ImageDescriptionHandle)
                                NewHandleClear(sizeof(ImageDescription));

    if (mySampleDesc == NIL)
        return(NIL);
#endif
}
```



```
// fill in the fields of the sample description
(**mySampleDesc).cType = theEffectType;
(**mySampleDesc).idSize = sizeof(ImageDescription);
(**mySampleDesc).hRes = 72L << 16;
(**mySampleDesc).vRes = 72L << 16;
(**mySampleDesc).frameCount = 1;
(**mySampleDesc).depth = 0;
(**mySampleDesc).clutID = -1;
#endif

(**mySampleDesc).vendor = kAppleManufacturer;
(**mySampleDesc).temporalQuality = codecNormalQuality;
(**mySampleDesc).spatialQuality = codecNormalQuality;
(**mySampleDesc).width = theWidth;
(**mySampleDesc).height = theHeight;

return(mySampleDesc);
}
```

Version Notes

Introduced in QuickTime 4. Image descriptions built using sample code from earlier versions of QuickTime cannot be used when stacking effects.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Creating an Effect Sample Description” (V–2898)

MakeImageDescriptionForPixMap

Fills out an `ImageDescription` (IV–2285) structure corresponding to a `PixMap` (IV–2332) structure.

```
OSErr MakeImageDescriptionForPixMap (
    PixMapHandle      pixmap,
    ImageDescriptionHandle *idh );
```

`pixmap`

A handle to a `PixMap` (IV–2332) structure.

MakeMediaTimeTable

idh

The handle of an `ImageDescription` (IV–2285) structure. On entry, this parameter normally points to an `ImageDescription` structure whose contents are NIL. On return, the structure is correctly filled out for the selected `PixMap`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Related Java Methods

`quicktime.std.image.ImageDescription.ImageDescription()`

MakeMediaTimeTable

Returns a time table for the specified media.

```
OSErr MakeMediaTimeTable (
    Media          theMedia,
    long           **offsets,
    TimeValue      startTime,
    TimeValue      endTime,
    TimeValue      timeIncrement,
    short          firstDataRefIndex,
    short          lastDataRefIndex,
    long           *retdataRefSkew );
```

theMedia

The media for this operation. Your application obtains this identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

offsets

A handle to an unlocked relocatable memory block allocated by your application. The function returns the time table for the media in this block.

startTime

The first point of the media to be included in the time table. This time value is expressed in the media’s time coordinate system.

endTime

The last point of the media to be included in the time table. This time value is expressed in the media’s time coordinate system.

`timeIncrement`

The resolution of the time table. The values in a time table are for a points in the media, and these points are separated by the amount of time specified by this parameter. The time value is expressed in the media's time coordinate system.

`firstDataRefIndex`

An index to the first data reference for the media to be included in the time table. Set this parameter to -1 to include all data references for the media. Set this parameter to 1 to specify the first data reference for the media.

`lastDataRefIndex`

An index to the last data reference for the media to be included in the time table. The value 1 specifies the first data reference for the media. If the value of the `firstDataRefIndex` parameter is -1, set this parameter to 0.

`retdataRefSkew`

The offset to the next row of the time table, in long integers. The next row contains values for the next data reference, as explained below. By adding the value of this parameter to an offset into the table, you get the offset to the corresponding point for the next data reference.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See "Error Codes" (IV-2663).

Discussion

Your application must allocate an unlocked relocatable memory block for the time table to be returned and pass a handle to it in the `offsets` parameter. The `MakeMediaTimeTable` (II-788) function resizes the block to accommodate the time table it returns.

This time table is a two-dimensional array of long integers, organized so that each row in the table contains values for one data reference. The first column in the table contains values for the time in the media specified by the `startTime` parameter, and each subsequent column contains values for the point in the media that is later by the value specified by the `timeIncrement` parameter. Each long integer value in the table specifies the offset, in bytes, from the beginning of the data reference for that point in the media. The number of columns in the table is equal to $(\text{endTime} - \text{startTime}) / \text{timeIncrement}$, rounded up. Because of alignment issues, this value is not always the same as the value of the `retdataRefSkew` parameter.

Special Considerations

When all the data for a movie has been transferred, your application must dispose of the time table created by this function.

MakeThumbnailFromPicture

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Low-Level Download Control” (V-2772)

MakeThumbnailFromPicture

Creates a thumbnail picture from a specified `Picture` (IV-2331) structure.

```
OSErr MakeThumbnailFromPicture (
    PicHandle          picture,
    short              colorDepth,
    PicHandle          thumbnail,
    ICMProgressProcRecordPtr progressProc );
```

`picture`

A handle to the image from which the thumbnail is to be extracted. The image must be stored in a `Picture` (IV-2331) structure.

`colorDepth`

The depth at which the image is likely to be viewed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

`thumbnail`

A handle to the destination `Picture` (IV-2331) structure for the thumbnail image. The compressor resizes this handle for the resulting data.

`progressProc`

A pointer to an `ICMProgressProcRecord` (IV-2284) structure. During the operation, the Image Compression Manager will occasionally call a function to report its progress. You can provide a function through this structure. If you have not provided a **progress function**, set this parameter to `NIL`. If you pass a value of -1, you obtain a standard progress function.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Making Thumbnail Pictures” (V-2791)

Related Java Methods

`quicktime.qd.Pict.makeThumbnail()`

MakeThumbnailFromPictureFile

Creates a thumbnail picture from a specified **picture file**.

```
OSErr MakeThumbnailFromPictureFile (
    short          refNum,
    short          colorDepth,
    PicHandle      thumbnail,
    ICMProgressProcRecordPtr progressProc );
```

refNum

A file reference number for the PICT file from which the thumbnail is to be extracted.

colorDepth

The depth at which the image is likely to be viewed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

thumbnail

A handle to the destination picture structure for the thumbnail image. The compressor resizes this handle for the resulting data.

progressProc

A pointer to an `ICMProgressProcRecord` (IV-2284) structure. During the operation, the Image Compression Manager will occasionally call a function to report its progress. You can provide a function through this structure.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Making Thumbnail Pictures” (V-2791)

MakeThumbnailFromPixMap

MakeThumbnailFromPixMap

Creates a thumbnail picture from a specified `PixMap` (IV–2332) structure.

```
OSErr MakeThumbnailFromPixMap (
    PixMapHandle      src,
    const Rect        *srcRect,
    short             colorDepth,
    PicHandle         thumbnail,
    ICMProgressProcRecordPtr progressProc );
```

`src`

A handle to the image from which the thumbnail is to be extracted. The image must be stored in a `PixMap` (IV–2332) structure.

`srcRect`

A pointer to a `Rect` (IV–2399) structure that defines the portion of the image to use for the thumbnail.

`colorDepth`

The depth at which the image is likely to be viewed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

`thumbnail`

A handle to the destination picture structure for the thumbnail image. The compressor resizes this handle for the resulting data.

`progressProc`

A pointer to an `ICMProgressProcRecord` (IV–2284) structure. During the operation, the Image Compression Manager will occasionally call a function to report its progress. You can provide a function through this structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Making Thumbnail Pictures” (V–2791)

Related Java Methods

```
quicktime.qd.Pict.thumbnailFromQDGraphics(),
quicktime.qd.QDGraphics.makeThumbnail()
```

MakeTrackTimeTable

Returns a time table for a specified track in a movie.

```
OSErr MakeTrackTimeTable (
    Track          trackH,
    long           **offsets,
    TimeValue      startTime,
    TimeValue      endTime,
    TimeValue      timeIncrement,
    short          firstDataRefIndex,
    short          lastDataRefIndex,
    long           *retdataRefSkew );
```

trackH

The track for the operation. Your application gets this identifier from such functions as `NewMovieTrack` (II-1092) and `GetMovieTrack` (I-497).

offsets

A handle to an unlocked relocatable memory block allocated by your application. The function returns the time table for the track in this block.

startTime

The first point of the track to be included in the time table. This time value is expressed in the movie's time coordinate system.

endTime

The last point of the track to be included in the time table. This time value is expressed in the movie's time coordinate system.

timeIncrement

The resolution of the time table. The values in a time table are for a points in the track, and these points are separated by the amount of time specified by this parameter. The time value is expressed in the movie's time coordinate system.

firstDataRefIndex

An index to the first data reference for the track to be included in the time table. Set this parameter to -1 to include all data references for the track. Set this parameter to 1 to specify the first data reference for the track.

lastDataRefIndex

An index to the last data reference for the track to be included in the time table. The value 1 specifies the first data reference for the track. If the value of the `firstDataRefIndex` parameter is -1, set this parameter to 0.



MapMatrix

`retdataRefSkew`

The offset to the next row of the time table, as a long integer. The next row contains values for the next data reference, as explained below. By adding the value of this parameter to an offset into the table, you get the offset to the corresponding point for the next data reference.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

Your application must allocate an unlocked relocatable memory block for the time table to be returned and pass a handle to it in the `offsets` parameter. The `MakeTrackTimeTable` (II-793) function resizes the block to accommodate the time table it returns.

This time table is a two-dimensional array of long integers that is organized so that each row in the table contains values for one data reference. The first column in the table contains values for the time in the track specified by the `startTime` parameter, and each subsequent column contains values for the point in the track that is later by the value specified by the `timeIncrement` parameter. Each long integer value in the table specifies the offset, in bytes, from the beginning of the data reference for that point in the track. The number of columns in the table is equal to $(\text{endTime} - \text{startTime}) / \text{timeIncrement}$, rounded up. Because of alignment issues, this value is not always the same as the value of the `retdataRefSkew` parameter. If there are track edits for a track, they are reflected in the track’s time table.

Special Considerations

When all the data for a movie has been transferred, your application must dispose of the time table created by this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Low-Level Download Control” (V-2772)

MapMatrix

Alters an existing matrix so that it defines a transformation from one rectangle to another.


```
void MapMatrix (
    MatrixRecord    *matrix,
    const Rect      *fromRect,
    const Rect      *toRect );
```

matrix

A pointer to a matrix structure. The `MapMatrix` function modifies this matrix so that it performs a transformation in the rectangle specified by the `toRect` parameter that is analogous to the transformation it currently performs in the rectangle specified by the `fromRect` parameter.

fromRect

A pointer to the source `Rect` (IV–2399) structure.

toRect

A pointer to the destination `Rect` (IV–2399) structure.

Discussion

`MapMatrix` affects only the scaling and translation attributes of the matrix.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Matrices” (V–2764)

Related Java Methods

`quicktime.std.image.Matrix.map()`

See Also

This function is similar to `RectMatrix` (III–1451), with the exception that it concatenates the translation and scaling operations to the previous contents of the matrix, whereas `RectMatrix` first sets the matrix to the identity state.

MCActivate

Lets a controller respond to activate, deactivate, suspend, and resume events.

```
ComponentResult MCActivate (
    MovieController    mc,
    WindowPtr          w,
    Boolean             activate );
```

MCAdjustCursor

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

w

A pointer to the window in which the event has occurred.

activate

The nature of the event. Set this parameter to `TRUE` for activate and resume events. Set it to `FALSE` for deactivate and suspend events.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Customizing Event Processing” (V–2779)

Related Java Methods

`quicktime.std.movies.MovieController.activate()`

MCAdjustCursor

Undocumented

```
ComponentResult MCAdjustCursor (
    MovieController    mc,
    WindowPtr          w,
    Point              where,
    long               modifiers );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

w

A pointer to the window in which the cursor is located.

where

The location of the cursor. This value is expressed in the local coordinates of the window specified by the *w* parameter.

modifiers

The cursor form (see below).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

modifiers Constants

`kQTCursorOpenHand`

Open hand cursor.

`kQTCursorClosedHand`

Closed hand cursor.

`kQTCursorPointingHand`

Pointing hand cursor.

`kQTCursorRightArrow`

Right arrow cursor.

`kQTCursorLeftArrow`

Left arrow cursor.

`kQTCursorDownArrow`

Down arrow cursor.

`kQTCursorUpArrow`

Up arrow cursor.

`kQTCursorIBeam`

I-beam cursor.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

MCAddMovieSegment

Undocumented

```
ComponentResult MCAddMovieSegment (
    MovieController mc,
```

MCClear

```
Movie          srcMovie,  
Boolean        scaled );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

srcMovie

The source movie. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), or `NewMovieFromHandle` (II–1084).

scaled

Undocumented

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `Movies.h`

MCClear

Removes the current movie selection from the movie associated with a specified controller.

```
ComponentResult MCClear (  
    MovieController mc );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Movies With a Controller” (V-2777)

Related Java Methods

`quicktime.std.movies.MovieController.clear()`

MCClick

Lets a controller respond when the user clicks in a movie controller window.

```
ComponentResult MCClick (
    MovieController    mc,
    WindowPtr         w,
    Point              where,
    long               when,
    long               modifiers );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131), or from `NewMovieController` (II-1071).

w

A pointer to the window in which the event has occurred.

where

The location of the click. This value is expressed in the local coordinates of the window specified by the *w* parameter. Your application must convert this value from the global coordinates returned in the `EventRecord` (IV-2246) structure.

when

Indicates when the user pressed the mouse button. You obtain this value from the `EventRecord` (IV-2246) structure.

modifiers

Specifies modifier flags for the event. You obtain this value from the `EventRecord` (IV-2246) structure.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

MCCopy

Programming Info

C interface file: `Movies.h`

Programming summary: “Customizing Event Processing” (V–2779)

Related Java Methods

`quicktime.std.movies.MovieController.click()`

MCCopy

Returns a copy of the current movie selection from the movie associated with a specified controller.

```
Movie MCCopy (  
    MovieController mc );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

function result A copy of the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Movies With a Controller” (V–2777)

Related Java Methods

`quicktime.std.movies.MovieController.copy()`

MCCut

Returns a copy of the current movie selection from the movie associated with a specified controller and then removes the current movie selection from the source movie.

```
Movie MCCut (  
    MovieController mc );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131), or from `NewMovieController` (II-1071).

function result A copy of the current movie selection.

Discussion

Your application is responsible for the returned movie. `MCCut` returns a movie containing the current selection from the movie associated with the specified controller. If the user has not made a selection, the returned movie reference is set to `NIL`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Movies With a Controller” (V-2777)

Related Java Methods

`quicktime.std.movies.MovieController.cut()`

MCDoAction

Invokes a movie controller component and makes it perform a specified action.

```
ComponentResult MCDoAction (
    MovieController mc,
    short          action,
    void           *params );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131), or from `NewMovieController` (II-1071).

action

The action to be taken. See “Movie Controller Actions” (V-2773).

params

A pointer to the parameter data appropriate to the action. See “Movie Controller Actions” (V-2773) for information about the parameters required for each supported action.

function result You can access Movie Toolbox error returns through `GetMoviesError`

MCDoAction

(I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Movie Controller Actions” (V-2773)

Related Java Methods

```
quicktime.std.movies.MovieController.activate(),
quicktime.std.movies.MovieController.deactivate(),
quicktime.std.movies.MovieController.play(),
quicktime.std.movies.MovieController.goToTime(),
quicktime.std.movies.MovieController.setVolume(),
quicktime.std.movies.MovieController.getVolume(),
quicktime.std.movies.MovieController.step(),
quicktime.std.movies.MovieController.setLooping(),
quicktime.std.movies.MovieController.getLooping(),
quicktime.std.movies.MovieController.setLoopIsPalindrome(),
quicktime.std.movies.MovieController.getLoopIsPalindrome(),
quicktime.std.movies.MovieController.setGrowBoxBounds(),
quicktime.std.movies.MovieController.setSelectionBegin(),
quicktime.std.movies.MovieController.setSelectionDuration(),
quicktime.std.movies.MovieController.setKeysEnabled(),
quicktime.std.movies.MovieController.getKeysEnabled(),
quicktime.std.movies.MovieController.setPlaySelection(),
quicktime.std.movies.MovieController.getPlaySelection(),
quicktime.std.movies.MovieController.setUseBadge(),
quicktime.std.movies.MovieController.getUseBadge(),
quicktime.std.movies.MovieController.setFlags(),
quicktime.std.movies.MovieController.getFlags(),
quicktime.std.movies.MovieController.setPlayEveryFrame(),
quicktime.std.movies.MovieController.getPlayEveryFrame(),
quicktime.std.movies.MovieController.getPlayRate(),
quicktime.std.movies.MovieController.suspend(),
quicktime.std.movies.MovieController.resume(),
quicktime.std.movies.MovieController.setControllerKeysEnabled(),
quicktime.std.movies.MovieController.getTimeSliderRect(),
quicktime.std.movies.MovieController.getDragEnabled(),
quicktime.std.movies.MovieController.setDragEnabled(),
quicktime.std.movies.MovieController.getSelectionBegin(),
```



```

quicktime.std.movies.MovieController.getSelectionDuration(),
quicktime.std.movies.MovieController.prerollAndPlay(),
quicktime.std.movies.MovieController.getCursorSettingEnabled(),
quicktime.std.movies.MovieController.setCursorSettingEnabled(),
quicktime.std.movies.MovieController.setColorTable(),
quicktime.std.movies.MovieController.controllerSizeChanged(),
quicktime.std.movies.MovieController.badgeClick(),
quicktime.std.movies.MovieController.movieEdited(),
quicktime.std.movies.MovieController.forceTimeTableUpdate(),
quicktime.std.movies.MovieController.linkToURL(),
quicktime.std.movies.MovieController.clickAndHoldPoint()

```

MCDraw



Responds to an update event.

```

ComponentResult MCDraw (
    MovieController mc,
    WindowPtr      w );

```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

w

A pointer to the window in which the update event has occurred.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Customizing Event Processing” (V–2779)

Related Java Methods

`quicktime.std.movies.MovieController.draw()`

MCDrawBadge

MCDrawBadge

Displays a controller's badge.

```
ComponentResult MCDrawBadge (  
    MovieController    mc,  
    RgnHandle          movieRgn,  
    RgnHandle          *badgeRgn );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

movieRgn

The boundary region of the controller's movie.

badgeRgn

A pointer to a region that is to receive information about the location of the badge. The movie controller returns the region where the badge is displayed. If you are not interested in this information, you may set this parameter to `NIL`. Your application must dispose of this handle.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

This function places the badge in an appropriate location based on the location of the controller's movie. `MCDrawBadge` can be useful in circumstances where you are using a movie controller component but do not want to incur the overhead of having the QuickTime movie in memory all the time. This function allows you to display the badge without having to display the movie. In addition, you can use the badge region to perform mouse-down event testing.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

```
quicktime.qd.Region.fromMovieControllerBadge(),  
quicktime.std.movies.MovieController.drawBadge()
```

MCEnableEditing

Enables and disables editing for a movie controller.

```
ComponentResult MCEnableEditing (
    MovieController    mc,
    Boolean             enabled );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

enabled

Specifies whether to enable or disable editing for the controller. Set this parameter to `TRUE` to enable editing; set it to `FALSE` to disable editing.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Once editing is enabled for a controller, the user may edit the movie associated with the controller.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Movies With a Controller” (V–2777)

Related Java Methods

`quicktime.std.movies.MovieController.enableEditing()`

MCGetClip

Obtains information describing a movie controller’s clipping regions.

```
ComponentResult MCGetClip (
    MovieController    mc,
    RgnHandle          *theClip,
    RgnHandle          *movieClip );
```

MCGetControllerBoundsRect

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

theClip

A pointer to a field that is to receive a handle to the clipping region of the entire movie controller. You must dispose of this region when you are done with it. If you are not interested in this information, set this parameter to `NIL`.

movieClip

A pointer to a field that is to receive a handle to the clipping region of the controller’s movie. You must dispose of this region when you are done with it. If you are not interested in this information, set this parameter to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

`quicktime.qd.Region.fromMovieControllerClip()`,
`quicktime.std.movies.MovieController.getClip()`

See Also

For the corresponding set function, see `MCSetClip` (II–830).

MCGetControllerBoundsRect

Returns a movie controller’s boundary rectangle.

```
ComponentResult MCGetControllerBoundsRect (  
    MovieController    mc,  
    Rect                *bounds );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

bounds

A pointer to a Rect (IV–2399) structure that is to receive the coordinates of the movie controller’s boundary rectangle. If there is insufficient screen space to display the controller, the function may return an empty structure.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

quicktime.std.movies.MovieController.getBounds()

See Also

For the corresponding set function, see MCSetControllerBoundsRect (II–832).

MCGetControllerBoundsRgn

Returns the actual region occupied by the controller and its movie.

```
RgnHandle MCGetControllerBoundsRgn (  
    MovieController mc );
```

mc

The movie controller for the operation. You obtain this identifier from OpenComponent (II–1130) or OpenDefaultComponent (II–1131), or from NewMovieController (II–1071).

function result A handle to a MacRegion (IV–2303) structure that reflects the size, shape, and location of the controller. Your application must dispose of this structure.

Discussion

As with MCGetControllerBoundsRect (II–806), this function returns a region even if the controller is hidden. Some movie controllers may not be rectangular in shape. If the movie is not attached to its controller, the boundary region encloses only the control portion of the controller.

MCGetControllerInfo

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

`quicktime.qd.Region.fromMovieControllerBounds()`,
`quicktime.std.movies.MovieController.getBoundsRgn()`

See Also

The rectangle returned by `MCGetControllerBoundsRect` (II–806) bounds the region returned by this function.

MCGetControllerInfo

Determines the current status of a movie controller and its associated movie, for menu highlighting.

```
ComponentResult MCGetControllerInfo (  
    MovieController mc,  
    long *someFlags );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

someFlags

A pointer to flags (see below) that specify the current status and capabilities of the controller. More than one flag may be set to 1.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

someFlags Constants

`mcInfoUndoAvailable`

The user has edited the movie. If this flag is set to 1, you can call `MCUndo` (II–838).

`mcInfoCutAvailable`

The user has selected some material in the movie and editing is enabled. If this flag is set to 1, you can call `MCCut` (II–800).

`mcInfoCopyAvailable`

The user has selected some material in the movie. If this flag is set to 1, you can call `MCCopy` (II-800).

`mcInfoPasteAvailable`

There is movie data in the scrap and editing is enabled. If this flag is set to 1, you can call `MCPaste` (II-824). If your application maintains a private scrap, this flag does not reflect the state of that scrap.

`mcInfoClearAvailable`

The user has selected some material in the movie and editing is enabled. If this flag is set to 1, you can call `MCClear` (II-798).

`mcInfoHasSound`

The movie has sound. If this flag is set to 1, the controller can play a movie's sound.

`mcInfoIsPlaying`

If this flag is set to 1, the movie is playing.

`mcInfoIsLooping`

The controller is currently set to play its movie repeatedly. If this flag is set to 1, the movie is looping.

`mcInfoIsInPalindrome`

The controller is currently set to play its movie repeatedly, alternating between forward and backward playback. If this flag is set to 1, the movie is in palindrome looping mode.

`mcInfoEditingEnabled`

The user can edit the movie associated with this controller. If this flag is set to 1, you have enabled editing by calling `MCEnableEditing` (II-805).

Discussion

You can use the information returned by this function to control your application's menu highlighting.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Handling Movie Events" (V-2776)

Related Java Methods

`quicktime.std.movies.MovieController.getControllerInfo()`

MCGetControllerPort

MCGetControllerPort

Returns a movie controller's color graphics port.

```
CGrafPtr MCGetControllerPort (  
    MovieController    mc );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

function result A pointer to the movie controller's `CGrafPort` (IV–2168) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

```
quicktime.std.movies.MovieController.getPort(),  
quicktime.qd.QDGraphics.fromMovieController()
```

See Also

For the corresponding set function, see `MCSetControllerPort` (II–833).

MCGetCurrentTime

Obtains the time value represented by the indicator on the movie controller's slider.

```
TimeValue MCGGetCurrentTime (  
    MovieController    mc,  
    TimeScale          *scale );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

scale

A pointer to a field that is to receive the time scale for the controller.

function result A time value containing the time shown by the indicator on the

movie controller's slider.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Getting and Setting Movie Controller Time” (V-2778)

Related Java Methods

`quicktime.std.movies.MovieController.getCurrentTime()`,
`quicktime.std.movies.MovieController.getTimeScale()`

MCGetDoActionsProc

Retrieves the `DoMCActionProc` (III-2082) callback attached to a movie controller.

```
ComponentResult MCGetDoActionsProc (
    MovieController mc,
    DoMCActionUPP *doMCActionProc,
    long *doMCActionRefCon );
```

`mc`

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131), or from `NewMovieController` (II-1071).

`doMCActionProc`

A pointer to a `DoMCActionProc` (III-2082) callback.

`doMCActionRefCon`

A reference constant that is passed to your callback. This parameter may point to a data structure containing information your function needs.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

MCGetIndMovie

MCGetIndMovie

Lets your application to retrieve the movie reference for a movie that is associated with a movie controller.

```
Movie MCGetIndMovie (
    MovieController    mc,
    short              index );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

index

Index for the movie. When set to 0, this call duplicates the action of the previous call to this function.

function result The movie identifier for the movie that is assigned to the specified controller, or NIL if there is no movie assigned to the controller.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Associating Movies With Controllers” (V–2774)

Related Java Methods

```
quicktime.std.movies.MovieController.getMovie(),
quicktime.std.movies.MultiMovieController.getIndMovie()
```

MCGetInterfaceElement

Gets the interface element of a specified type for a movie controller.

```
ComponentResult MCGetInterfaceElement (
    MovieController    mc,
    MCInterfaceElement whichElement,
    void               *element );
```

`mc`

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

`whichElement`

A constant (see below) that identifies the interface element type.

`element`

A pointer to the element type.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

whichElement Constants

`kMCIEEnabledButtonPicture`

A Picture (IV–2331) of an enabled button.

`kMCIEDisabledButtonPicture`

A Picture (IV–2331) of a disabled button.

`kMCIEDepressedButtonPicture`

A Picture (IV–2331) of a depressed button.

`kMCIEEnabledSizeBoxPicture`

A Picture (IV–2331) of an enabled size box.

`kMCIEDisabledSizeBoxPicture`

A Picture (IV–2331) of a disabled size box.

`kMCIEEnabledUnavailableButtonPicture`

A Picture (IV–2331) of an enabled but unavailable button.

`kMCIEDisabledUnavailableButtonPicture`

A Picture (IV–2331) of a disabled and unavailable button.

`kMCIESoundSlider`

The sound slider.

`kMCIESoundThumb`

The indicator on the sound slider.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

MCGetMenuString

MCGetMenuString

Retrieves the text string for a movie controller menu command.

```
ComponentResult MCGetMenuString (  
    MovieController    mc,  
    long               modifiers,  
    short              item,  
    Str255              aString );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

modifiers

The current modifiers from the mouse-down or key-down event to which you are responding.

item

One of the movie controller Edit menu constants (see below).

aString

On entry, pass a string of type `Str255`; on exit, this string is set to the text of the menu item specified by the *item* parameter.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

item Constants

mcMenuUndo

Return the menu item text for the Undo command.

mcMenuCut

Return the menu item text for the Cut command.

mcMenuCopy

Return the menu item text for the Copy command.

mcMenuPaste

Return the menu item text for the Paste command.

mcMenuClear

Return the menu item text for the Clear command.

Discussion

`MCGetMenuString` is used by `MCSetUpEditMenu` (II–836).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Movies With a Controller” (V-2777)

See Also

See `MCSetUpEditMenu` (II-836).

MCGetVisible

Returns a value that indicates whether or not a movie controller is visible.

```
ComponentResult MCGetVisible (
    MovieController mc );
```

`mc`

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131), or from `NewMovieController` (II-1071).

function result If the controller is visible, the function result is set to 1. If the controller is not showing, the function result is set to 0. You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494). See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V-2775)

Related Java Methods

`quicktime.std.movies.MovieController.getVisible()`

See Also

For the corresponding set function, see `MCSetVisible` (II-837).

MCGetWindowRgn

Determines the window region that is actually in use by a controller and its movie.

MCIdle

```
RgnHandle MCGetWindowRgn (  
    MovieController    mc,  
    WindowPtr          w );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

w

A pointer to the window in which the movie controller and its movie are displayed, if the control portion of the controller is attached to the movie. If the controller is detached and in a separate window from the movie, specify one of the windows.

function result A handle to the `MacRegion` (IV–2303) structure for the window that is actually in use. Your application must dispose of this structure.

Discussion

The region returned by this function contains only the visible portions of the controller and its movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

```
quicktime.qd.Region.fromMovieControllerWindow(),  
quicktime.std.movies.MovieController.getWindowRgn()
```

MCIdle

Performs idle processing for a movie controller.

```
ComponentResult MCIdle (  
    MovieController    mc );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Customizing Event Processing” (V-2779)

Related Java Methods

`quicktime.std.movies.MovieController.idle()`

MCInvalidate

Invalidates a region of a movie controller’s display.

```
ComponentResult MCInvalidate (
    MovieController mc,
    WindowPtr      w,
    RgnHandle       invalidRgn );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131), or from `NewMovieController` (II-1071).

w

A pointer to the window in which the movie controller and its movie are displayed, if the control portion of the controller is attached to the movie. If the controller is detached and in a separate window from the movie, specify one of the windows.

invalidRgn

A handle to a `MacRegion` (IV-2303) structure that defines a region to invalidate.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

MCIsControllerAttached

Related Java Methods

```
quicktime.std.movies.MovieController.invalidate()
```

MCIsControllerAttached

Returns a value that indicates whether a movie controller is attached to its movie.

```
ComponentResult MCIsControllerAttached (  
    MovieController mc );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

function result If the controller is attached, the returned value is set to 1. If the controller is not attached, the returned value is set to 0. You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

```
quicktime.std.movies.MovieController.isAttached()
```

See Also

You can use `MCSetControllerAttached` (II–831) to attach or detach a movie controller.

MCIsEditingEnabled

Determines whether editing is currently enabled for a movie controller.

```
long MCIsEditingEnabled (  
    MovieController mc );
```


mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

function result Returns 1 if editing is enabled; set to 0 if editing is disabled or if the controller component does not support editing.

Discussion

Once editing is enabled for a controller, the user may edit the movie associated with the controller.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Movies With a Controller” (V–2777)

Related Java Methods

`quicktime.std.movies.MovieController.isEditingEnabled()`

MCIsPlayerEvent

Handles all events for a movie controller.

```
ComponentResult MCIsPlayerEvent (
    MovieController    mc,
    const EventRecord  *e );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

e

A pointer to the current `EventRecord` (IV–2246) structure.

function result A long integer indicating whether the movie controller component handled the event. The component sets this long integer to 1 if it handled the event. Your application should then skip the rest of its event loop and wait for the next event. The return value is 0 otherwise. Your application must then handle the event as part of its normal event processing.

MCIsPlayerEvent

Discussion

The movie controller component does everything necessary to support the movie controller and its associated movie. For example, the component calls `MoviesTask` (II-973) for each movie. The movie controller component also handles suspend and resume events. It treats suspend events as deactivate requests and resume events as activate requests.

The following sample code shows how to convert Windows messages to Macintosh events and then pass those events to the QuickTime movie controller, using this function:

```
// MCIsPlayerEvent coding example
// See "Discovering QuickTime," page 240
MovieController mc; // Movie controller for movie

LRESULT CALLBACK WndProc
    (HWND      hwnd, // Handle to window
     UINT      iMsg, // Message type
     WPARAM    wParam, // Message-dependent parameter
     LPARAM    lParam) // Message-dependent parameter
{
    MSG        msg; // Windows message structure
    EventRecord er; // Macintosh event record
    DWORD      dwPos; // Mouse coordinates of message
    msg.hwnd    = hwnd; // Window handle
    msg.message = iMsg; // Message type
    msg.wParam  = wParam; // Word-length parameter
    msg.lParam  = lParam; // Long-word parameter

    msg.time = GetMessageTime(); // Get time of message
    dwPos = GetMessagePos(); // Get mouse position
    msg.pt.x = LOWORD(dwPos); // Extract x coordinate
    msg.pt.y = HIWORD(dwPos); // Extract y coordinate

    WinEventToMacEvent(&msg, &er); // Convert to event
    MCIsPlayerEvent(mc, &er); // Pass event to QuickTime

    switch (iMsg) { // Dispatch on message type

        . . . // Handle message according to type
```

```
        } // end switch (iMsg)

    } // end WndProc
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Handling Movie Events” (V–2776)

See Also

You can provide an action filter function that is called by the movie controller component. The component calls your filter function after it decides to process a particular action, but before it actually does so. In this manner, your application can perform custom action processing for a movie controller. Set your action filter function with `MCSetActionFilterWithRefCon` (II–829).



MCKey

Handles keyboard events for a movie controller.

```
ComponentResult MCKey (
    MovieController mc,
    SInt8          key,
    long           modifiers );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

key

The keystroke. You obtain this value from the event structure.

modifiers

Modifier flags for the event. You obtain this value from the `EventRecord` (IV–2246) structure.

function result

You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

MCMovieChanged

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Customizing Event Processing” (V–2779)

Related Java Methods

`quicktime.std.movies.MovieController.key()`

MCMovieChanged

Informs a movie controller component that your application has used the Movie Toolbox to change the characteristics of its associated movie.

```
ComponentResult MCMovieChanged (  
    MovieController    mc,  
    Movie              m );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

m

The movie that has been changed.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Controller Actions” (V–2773)

Related Java Methods

`quicktime.std.movies.MovieController.movieChanged()`,

`quicktime.std.movies.MultiMovieController.movieChanged()`

MCNewAttachedController

Associates a specified movie with a movie controller.

```
ComponentResult MCNewAttachedController (
    MovieController    mc,
    Movie              theMovie,
    WindowPtr          w,
    Point               where );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131), or from `NewMovieController` (II-1071).

theMovie

The movie to be associated with the movie controller.

w

A pointer to the window in which the movie is to be displayed. The movie controller component sets the movie's graphics world to match this window. If you set the *w* parameter to `NIL`, the component uses the current window.

where

The upper-left corner of the movie within the window specified by the *w* parameter. The movie controller component uses the movie's boundary `Rect` (IV-2399) structure to determine the size of the movie. `GetMovieBox` (I-460) returns this structure.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See "Error Codes" (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Associating Movies With Controllers" (V-2774)

Related Java Methods

```
quicktime.std.movies.MovieController.MovieController(),
quicktime.std.movies.MultiMovieController.MultiMovieController()
```

MCPaste

MCPaste

Inserts a specified movie at the current movie time in the movie associated with a specified controller.

```
ComponentResult MCPaste (
    MovieController    mc,
    Movie              srcMovie );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

srcMovie

The movie to be inserted into the current selection in the movie associated with the movie controller specified by the *mc* parameter. If you set this parameter to `NIL`, the movie controller component retrieves the source movie from the scrap.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Movies With a Controller” (V–2777)

Related Java Methods

`quicktime.std.movies.MovieController.paste()`

MCPositionController

Controls the position of a movie and its controller on the computer display.

```
ComponentResult MCPositionController (
    MovieController    mc,
    const Rect         *movieRect,
    const Rect         *controllerRect,
    long               someFlags );
```

`mc`

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131), or from `NewMovieController` (II-1071).

`movieRect`

A pointer to a `Rect` (IV-2399) structure that specifies the coordinates of the movie's boundary `Rect` (IV-2399) structure.

`controllerRect`

A pointer to a `Rect` (IV-2399) structure that specifies the coordinates of the controller's boundary `Rect` (IV-2399) structure. The movie controller component always centers the control portion of the controller inside this rectangle. The movie controller component only uses this parameter when the control portion of the controller is detached from the movie. If you are working with an attached controller, you can set this parameter to `NIL`.

`someFlags`

Flags (see below) that control how the movie is drawn. If you set these flags to 0, the movie controller component centers the movie in the rectangle specified by `movieRect` and scales the movie to fit in that rectangle.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See "Error Codes" (IV-2663).

someFlags Constants

`mcTopLeftMovie`

If this flag is set to 1, the movie controller component places the movie into the upper-left corner of the display rectangle specified by the `movieRect` parameter. The component scales the movie to fit into the rectangle. Note that the control portion of the controller may fall outside of the rectangle, depending upon the results of the scaling operation.

`mcScaleMovieToFit`

If this flag is set to 1, the movie controller component resizes the movie to fit into the display rectangle specified by the `movieRect` parameter after it places the control portion of the controller into the rectangle. If you set this flag and the `mcTopLeftMovie` flag to 1, the movie controller component resizes the movie to fit into the specified rectangle and places the control portion of the controller outside of the rectangle.

`mcPositionDontInvalidate`

If this flag is set to 1, the movie controller component is requested not to invalidate areas of the window that are changed as a result of repositioning the



MCPtInController

movie or the controller. This flag is useful for applications that use the movie controller as part of a larger document. In particular, if the document is scrolled using the Mac OS ScrollRect function, optimal redrawing occurs (that is, scrolled areas are not redrawn) if this flag is set.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

quicktime.std.movies.MovieController.position()

MCPtInController

Reports whether a point is in the control area of a movie.

```
ComponentResult MCPtInController (
    MovieController    mc,
    Point              thePt,
    Boolean             *inController );
```

mc

The movie controller for the operation. You obtain this identifier from OpenComponent (II–1130) or OpenDefaultComponent (II–1131), or from NewMovieController (II–1071).

thePt

The point to be checked. This point must be passed in local coordinates to the controller’s window. This point is checked only against the movie controller’s controls, not the movie itself.

inController

Returns true if the point is in the controller; false if it is not.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

While you could always determine if a point is contained in a movie, using PtInMovie (II–1148), the MCPtInController function allows you to determine if a point is in the control area of a movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Handling Movie Events” (V–2776)

Related Java Methods

`quicktime.std.movies.MovieController.inController()`

MCRemoveAllMovies

Removes all movies associated with a controller.

```
ComponentResult MCRemoveAllMovies (
    MovieController mc );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.MultiMovieController.removeAllMovies()`

MCRemoveAMovie

Removes one movie from a multi-movie controller.

```
ComponentResult MCRemoveAMovie (
    MovieController mc,
    Movie m );
```

MCRemoveMovie

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), or `NewMovieFromHandle` (II–1084).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.MultiMovieController.removeAMovie()`

MCRemoveMovie

Removes a movie from a movie controller.

```
ComponentResult MCRemoveMovie (  
    MovieController mc );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

```
quicktime.std.movies.MovieController.removeMovie(),
quicktime.std.movies.MultiMovieController.removeMovie()
```

MCSetActionFilter

Sets the MCActionFilterProc (III–2096) callback for a movie controller.

```
ComponentResult MCSetActionFilter (
    MovieController    mc,
    MCActionFilterUPP  blob );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

blob

A **Universal Procedure Pointer** to an MCActionFilterProc (III–2096) callback.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

MCSetActionFilterWithRefCon

Establishes an action filter function for a movie controller.

```
ComponentResult MCSetActionFilterWithRefCon (
    MovieController    mc,
    MCActionFilterWithRefConUPP  blob,
    long               refCon );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

MCSetClip

blob

A pointer to your `MCActionFilterWithRefConProc` (III–2097) callback. Set this parameter to `NIL` to remove an existing callback.

refCon

A reference constant value. The movie controller component passes this reference constant to your action filter callback each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The movie controller component calls your action filter function each time the component receives an action for its movie controller. Your filter function is then free to handle the action or to refer it back to the movie controller component. If you refer it back to the movie controller component, the component handles the action.

If your filter function handles an action, you can handle the action in any way you desire. For example, your filter function could change the operation of movie controller buttons. More commonly, applications use the action filter function to monitor actions of the controller. For instance, your filter function might enable you to find out when the user clicks the play button, so that your application can enable appropriate menu selections. Alternatively, you can use the filter function to detect when the user resizes the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Controller Actions” (V–2773)

Related Java Methods

`quicktime.std.movies.MovieController.setActionFilter()`,
`quicktime.std.movies.MovieController.removeActionFilter()`

MCSetClip

Lets you set a movie controller’s clipping region.

`ComponentResult MCSetClip (`

```
MovieController    mc,
RgnHandle          theClip,
RgnHandle          movieClip );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

theClip

A handle to a region that defines the controller’s clipping region. This clipping region affects the entire movie controller and its movie, including the controller’s badge and associated controls. Set this parameter to `NIL` to clear the controller’s clipping region.

movieClip

A handle to a region that defines the clipping region of the controller’s movie. This clipping region affects only the movie and the badge, not the movie controller. Set this parameter to `NIL` to clear the movie clipping region.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

```
quicktime.std.movies.MovieController.setClip(),
quicktime.std.movies.MovieController.setMovieClip()
```

See Also

For the corresponding get function, see `MCGetClip` (II–805).

MCSetControllerAttached

Lets your application control whether a movie controller is attached to its movie or detached from it.

```
ComponentResult MCSetControllerAttached (
    MovieController    mc,
    Boolean            attach );
```

MCSetControllerBoundsRect

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

attach

The action for this function. Set the `attach` parameter to `TRUE` to cause the controller to be attached to its movie. Set this parameter to `FALSE` to detach the controller from its movie.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

`quicktime.std.movies.MovieController.setAttached()`

MCSetControllerBoundsRect

Lets you change the position and size of a movie controller.

```
ComponentResult MCSetControllerBoundsRect (  
    MovieController    mc,  
    const Rect         *bounds );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

bounds

A pointer to a `Rect` (IV–2399) structure that contains the new boundary `Rect` (IV–2399) structure for the movie controller.

function result See “Error Codes” (IV–2663). Returns a value of `controllerBoundsNotExact` if the boundary rectangle has been changed but does not correspond to the rectangle you specified. In this case, the new boundary rectangle is always smaller than the

requested rectangle. Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

`quicktime.std.movies.MovieController.setBounds()`

See Also

For the corresponding get function, see `MCGetControllerBoundsRect` (II–806).

MCSetControllerPort

Lets your application set the graphics port for a movie controller.

```
ComponentResult MCSetControllerPort (
    MovieController mc,
    CGrafPtr gp );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

gp

A pointer to the new graphics port for the movie controller. Set this parameter to `NIL` to use the current graphics port.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Movie controller components use `MCSetControllerPort` each time you create a new movie controller. Hence, your component must be set to a valid port before creating a new movie controller. You can use this function to place a movie and its associated movie controller in different graphics ports. If you are using an attached controller, both the controller and the movie’s graphics ports are changed. If you are using a detached controller, this function changes only the graphics port of the control portion of the controller. You must use `SetMovieGWorld` (III–1619) followed by `MCMovieChanged` (II–822) to change other portions.

MCSetDuration

```
pascal ComponentResult MCSetControllerPort (MovieController mc,  
                                           CGrafPtr gp);
```

Special Considerations

The movie controller component may use the foreground and background colors from the graphics port at the time this function is called to colorize the movie controller.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

`quicktime.std.movies.MovieController.setPort()`

See Also

For the corresponding get function, see `MCGetControllerPort` (II–810).

MCSetDuration

Lets your application set a controller’s duration in the case where a controller does not have a movie associated with it.

```
ComponentResult MCSetDuration (  
    MovieController    mc,  
    TimeValue          duration );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

duration

The new duration for the movie. This duration value must be in the controller’s time scale.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Getting and Setting Movie Controller Time” (V-2778)

Related Java Methods

`quicktime.std.movies.MovieController.setDuration()`

MCSetMovie

Associates a movie with a specified movie controller.

```
ComponentResult MCSetMovie (
    MovieController    mc,
    Movie              theMovie,
    WindowPtr          movieWindow,
    Point              where );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131), or from `NewMovieController` (II-1071).

theMovie

The movie to be associated with the movie controller. Set this value to `NIL` to remove the movie from the controller.

movieWindow

The window in which the movie is to be displayed. The movie controller component sets the movie’s graphics world to match this window. If you set the *w* parameter to `NIL`, the component uses the current window.

where

The upper-left corner of the movie within the window specified by the *movieWindow* parameter. The movie controller component uses the movie’s boundary `Rect` (IV-2399) structure to determine the size of the movie. `GetMovieBox` (I-460) returns this structure.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

MCSetUpEditMenu

Programming Info

C interface file: `Movies.h`

Programming summary: “Associating Movies With Controllers” (V-2774)

Related Java Methods

```
quicktime.std.movies.MovieController.setMovie(),  
quicktime.std.movies.MultiMovieController.setMovie(),  
quicktime.std.movies.MultiMovieController.addMovie()
```

MCSetUpEditMenu

Correctly highlights and names the items in your application’s Edit menu.

```
ComponentResult MCSetUpEditMenu (  
    MovieController    mc,  
    long               modifiers,  
    MenuHandle         mh );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131), or from `NewMovieController` (II-1071).

modifiers

The current modifiers from the mouse-down or key-down event to which you are responding.

mh

A menu handler to your current Edit menu. The first six items in your Edit menu should be the standard editing commands: Undo, a blank line, Cut, Copy, Paste, and Clear.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Movies With a Controller” (V-2777)

MCSetVisible

Lets your application control the visibility of a movie controller.

```
ComponentResult MCSetVisible (
    MovieController    mc,
    Boolean            visible );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

visible

Set to `TRUE` to cause the controller to be visible, or `FALSE` to make the controller invisible.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Controller Attributes” (V–2775)

Related Java Methods

`quicktime.std.movies.MovieController.setVisible()`

See Also

For the corresponding get function, see `MCGetVisible` (II–815).

MCTrimMovieSegment

Undocumented

```
ComponentResult MCTrimMovieSegment (
    MovieController    mc );
```

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

MCUndo

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `Movies.h`

MCUndo

Lets your application discard the effects of the most recent edit operation.

```
ComponentResult MCUndo (  
    MovieController mc );
```

`mc`

The movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Movies With a Controller” (V–2777)

Related Java Methods

`quicktime.std.movies.MovieController.undo()`

Media3DGetCameraAngleAspect

Deprecated.

```
ComponentResult Media3DGetCameraAngleAspect (  
    MediaHandler mh,  
    QTFloatSingle *fov,  
    QTFloatSingle *aspectRatioXToY );
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

Media3DGetCameraData

Deprecated.

```
ComponentResult Media3DGetCameraData (
    MediaHandler    mh,
    void            *cameraData );
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

Media3DGetCameraRange

Deprecated.

```
ComponentResult Media3DGetCameraRange (
    MediaHandler    mh,
    void            *tQ3CameraRange );
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

Media3DGetCurrentGroup

Deprecated.

```
ComponentResult Media3DGetCurrentGroup (
    MediaHandler    mh,
    void            *group );
```



Media3DGetNamedObjectList

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

Media3DGetNamedObjectList

Deprecated.

```
ComponentResult Media3DGetNamedObjectList (
    MediaHandler      mh,
    QTAtomContainer   *objectList );
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

```
quicktime.std.movies.media.ThreeDMediaHandler.getNamedObjectList(),
quicktime.std.movies.AtomContainer.fromThreeDMediaHandlerObject()
```

Media3DGetRendererList

Deprecated.

```
ComponentResult Media3DGetRendererList (
    MediaHandler      mh,
    QTAtomContainer   *rendererList );
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

```
quicktime.std.movies.media.ThreeDMediaHandler.getRendererList(),
quicktime.std.movies.AtomContainer.fromThreeDMediaHandlerRenderer()
```

Media3DGetViewObject

Deprecated.

```
ComponentResult Media3DGetViewObject (
    MediaHandler    mh,
    void            *tq3viewObject );
```

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

Media3DRotateNamedObjectTo

Deprecated.

```
ComponentResult Media3DRotateNamedObjectTo (
    MediaHandler    mh,
    char            *objectName,
    Fixed           xDegrees,
    Fixed           yDegrees,
    Fixed           zDegrees );
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

Media3DScaleNamedObjectTo

Deprecated.

```
ComponentResult Media3DScaleNamedObjectTo (
    MediaHandler    mh,
    char            *objectName,
    Fixed           xScale,
    Fixed           yScale,
    Fixed           zScale );
```

function result

Media3DSetCameraAngleAspect

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

Media3DSetCameraAngleAspect

Deprecated.

```
ComponentResult Media3DSetCameraAngleAspect (
    MediaHandler      mh,
    QTFloatSingle     fov,
    QTFloatSingle     aspectRatioXToY );
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

Media3DSetCameraData

Deprecated.

```
ComponentResult Media3DSetCameraData (
    MediaHandler      mh,
    void              *cameraData );
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

Media3DSetCameraRange

Deprecated.

```
ComponentResult Media3DSetCameraRange (
    MediaHandler      mh,
    void              *tQ3CameraRange );
```


Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

Media3DTranslateNamedObjectTo

Deprecated.

```
ComponentResult Media3DTranslateNamedObjectTo (
    MediaHandler    mh,
    char            *objectName,
    Fixed           x,
    Fixed           y,
    Fixed           z );
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

MediaChangedNonPrimarySource

Notifies a media handler of a change in the source of media data from another media handler.

```
ComponentResult MediaChangedNonPrimarySource (
    MediaHandler    mh,
    long            inputIndex );
```

mh

A reference to a media handler. You can obtain this reference from `GetMediaHandler` (I-432).

inputIndex

The ID of the entry in the media's input map to which the changed data corresponds.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

MediaCompare

Programming Info

C interface file: `MediaHandlers.h`

MediaCompare

Lets a media handler determine whether the Movie Toolbox should allow one track to be pasted into another.

```
ComponentResult MediaCompare (
    MediaHandler      mh,
    Boolean           *isOK,
    Media             srcMedia,
    ComponentInstance srcMediaComponent );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

isOK

A pointer to a Boolean value. Your media handler must set this value to TRUE if the source media and the media associated with the media handler have equivalent media settings, so that pasting the two together would cause no media information loss.

srcMedia

The source media for this operation.

srcMediaComponent

The source media component for this operation.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: "General Data Management" (V-2861)

MediaCurrentMediaQueuedData

Retrieves the timing of the current media in queued data.

```
ComponentResult MediaCurrentMediaQueuedData (
```

```
MediaHandler    mh,  
long            *milliSecs );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

milliSecs

A pointer to the number of milliseconds to the current data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`



MediaDisposeTargetRefCon

Undocumented

```
ComponentResult MediaDisposeTargetRefCon (  
    MediaHandler    mh,  
    long            targetRefCon );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

targetRefCon

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

MediaDoIdleActions

Forces a media handler to perform its idle-time actions.

```
ComponentResult MediaDoIdleActions (  
    MediaHandler    mh );
```

MediaEmptySampleCache

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I-432).

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `MediaHandlers.h`

MediaEmptySampleCache

Deletes any sample data that the media handler has cached.

```
ComponentResult MediaEmptySampleCache (  
    MediaHandler    mh,  
    long            sampleNum,  
    long            sampleCount );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I-432).

sampleNum

The ID of the first sample to delete.

sampleCount

The number of samples to delete. Passing `-1` means delete `sampleNum` and all samples after it.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This is an optional media handler function. Most developers will not need to call it.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `MediaHandlers.h`

MediaEnterEmptyEdit

Undocumented

```
ComponentResult MediaEnterEmptyEdit (
```

```
MediaHandler mh );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

MediaFlushNonPrimarySourceData

Flushes data that a media handler gets from another media handler.

```
ComponentResult MediaFlushNonPrimarySourceData (  
    MediaHandler mh,  
    long inputIndex );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

inputIndex

The ID of the entry in the media’s input map to which the flushed data corresponds.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

MediaForceUpdate

Forces a media update.

```
ComponentResult MediaForceUpdate (  
    MediaHandler mh,  
    long forceUpdateFlags );
```

MediaGetActionsForQTEvent

mh

A media handler. You can obtain this reference from GetMediaHandler (I-432).

forceUpdateFlags

Flags (see below) that define the update to be forced.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

forceUpdateFlags Constants

forceUpdateRedraw

Force a redraw.

forceUpdateNewBuffer

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

MediaGetActionsForQTEvent

Undocumented

```
ComponentResult MediaGetActionsForQTEvent (  
    MediaHandler      mh,  
    QTEventRecordPtr  event,  
    long              targetRefCon,  
    QTAtomContainer   *container,  
    QTAtom             *atom );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I-432).

event

A pointer to a QTEventRecord (IV-2353) structure.

targetRefCon

Undocumented

container

Undocumented

atom

Undocumented

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

MediaGetClock

Gets the clock component associated with a media.

```
ComponentResult MediaGetClock (
    MediaHandler      mh,
    ComponentInstance *clock );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I-432).

clock

A pointer to a clock component.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

MediaGetDrawingRgn

Specifies a portion of the screen that must be redrawn, defined in the movie’s display coordinate system.

```
ComponentResult MediaGetDrawingRgn (
    MediaHandler      mh,
    RgnHandle         *partialRgn );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from GetMediaHandler (I-432).

partialRgn

A pointer to a handle to a MacRegion (IV-2303) structure that defines the screen region to be redrawn, using the movie’s display coordinate system. Note that

MediaGetEffectiveSoundBalance

your component is responsible for disposing of this region once drawing is complete. Since the base media handler will use this region during redrawing, it is best to dispose of it when your component is closed.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The Movie Toolbox calls this function in order to determine what part of the screen needs to be redrawn. By default, theMovie Toolbox redraws the entire region that belongs to your component. If your component determines that only a portion of the screen has changed, and has indicated this to the toolbox by setting the `mPartialDraw` flag to 1 in the `flagsOut` parameter of the `MediaIdle` (II–874) function, the toolbox calls your component’s `MediaGetDrawingRgn` (II–849) function. Your component returns a region that defines the changed portion of the track’s display region.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “Managing Graphics Data” (V–2864)

MediaGetEffectiveSoundBalance

Gets the effective sound balance setting of a media handler.

```
ComponentResult MediaGetEffectiveSoundBalance (
    MediaHandler    mh,
    short           *balance );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

balance

A pointer to an integer. The Movie Toolbox returns the current balance setting of the media handler as a 16-bit, fixed-point value. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Valid balance values range from –1.0 to 1.0. Negative values emphasize the left sound channel, and positive values emphasize the right sound channel; a value of 0 specifies neutral balance.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `MediaHandlers.h`

MediaGetEffectiveVolume

Gets the effective volume setting for a media handler.

```
ComponentResult MediaGetEffectiveVolume (
    MediaHandler    mh,
    short           *volume );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I-432).

volume

The media's current volume setting. This value is represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer portion; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `MediaHandlers.h`

MediaGetErrorString

Undocumented

```
ComponentResult MediaGetErrorString (
    MediaHandler    mh,
    ComponentResult theError,
    Str255          errorString );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I-432).

MediaGetGraphicsMode

theError

An error identifier; see “Error Codes” (IV–2663).

errorString

A text string that describes the error.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `MediaHandlers.h`

MediaGetGraphicsMode

Obtains the graphics mode and blend color values currently in use by any media handler.

```
ComponentResult MediaGetGraphicsMode (
    MediaHandler    mh,
    long            *mode,
    RGBColor        *opColor );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

mode

A pointer to a long integer. The media handler returns the graphics mode currently in use by the media handler; see “Graphics Transfer Modes” (IV–2670).

opColor

A pointer to an `RGBColor` (IV–2403) structure. The Movie Toolbox returns the color currently in use by the media handler. This is the blend value for blends and the transparent color for transparent operations. The toolbox supplies this value to `QuickDraw` when you draw in `addPin`, `subPin`, `blend`, `transparent`, or `graphicsModeStraightAlphaBlend mode`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “Managing Graphics Data” (V–2864), “Video Media Handler Functions” (V–2755)

Related Java Methods

`quicktime.std.movies.media.VisualMediaHandler.getGraphicsMode()`

See Also

You can set the graphics mode and blend color of any media handler by calling `MediaSetGraphicsMode` (II–892).

MediaGetInvalidRegion

Gets the invalid region for a media handler’s current display.

```
ComponentResult MediaGetInvalidRegion (
    MediaHandler    mh,
    RgnHandle       rgn );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

rgn

A handle to a `MacRegion` (IV–2303) structure that defines an invalid region.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

MediaGetMediaInfo

Lets a derived media handler obtain the private data stored in its media.

```
ComponentResult MediaGetMediaInfo (
    MediaHandler    mh,
    Handle          h );
```

MediaGetMediaLoadState

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

h

A handle to storage containing your media handler’s proprietary information. Your media handler creates this private data when the Movie Toolbox calls your `MediaPutMediaInfo` (II–884) function. Do not dispose of this handle; it is owned by the Movie Toolbox.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if you store private data in your media.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V–2861)

MediaGetMediaLoadState

Queried by `GetMovieLoadState` (I–477) to help determine a movie’s load state.

```
ComponentResult MediaGetMediaLoadState (
    MediaHandler    mh,
    long            *mediaLoadState );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

mediaLoadState

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

`GetMovieLoadState` (I–477) queries any idling importers associated with a movie, checks if the movie is fast starting, and queries media handlers. The minimum load state of all of these is then considered to be the load state of the movie.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `MediaHandlers.h`

MediaGetName

Returns the name of the media type.

```
ComponentResult MediaGetName (
    MediaHandler    mh,
    Str255          name,
    long            requestedLanguage,
    long            *actualLanguage );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

name

The name of the media type; for example, the video media handler returns the string 'video'.

requestedLanguage

The language in which you want the name returned; see "Localization Codes" (IV-2673).

actualLanguage

A pointer to the actual language in which the name is returned; see "Localization Codes" (IV-2673)

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: "General Data Management" (V-2861)

MediaGetNextBoundsChange

Determines when a media causes a spatial change to a movie.

```
ComponentResult MediaGetNextBoundsChange (
    MediaHandler    mh,
    TimeValue       *when );
```

MediaGetNextStepTime

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

when

A pointer to a movie time value, which your media handler must set. Be sure to use the movie's time base. Use the current effective rate to determine the direction your media is playing. Set this value to -1 if there are no more changes in the specified direction.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if you change the shape of your media's spatial representation during playback. The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of `MediaSetHandlerCapabilities` (II-894).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: "Managing Graphics Data" (V-2864)

MediaGetNextStepTime

Searches for the next forward or backward step time from the given media time.

```
ComponentResult MediaGetNextStepTime (
    MediaHandler    mh,
    short           flags,
    TimeValue       mediaTimeIn,
    TimeValue       *mediaTimeOut,
    Fixed           rate );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

flags

Flags (see below) that specify search parameters.

`mediaTimeIn`

A time value that establishes the starting point for the search. This time value is in the media's time scale.

`mediaTimeOut`

The step time (the time of the next frame) calculated by the media handler. The media handler should return the first time value it finds that meets the search criteria specified in the `flags` parameter. This time value is in the media's time scale.

`rate`

The search direction. Negative values search backward from the starting point specified in the `mediaTimeIn` parameter. Other values cause a forward search.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

flags Constants

`nextTimeStep`

Set this flag to 1 to search for the next frame. The normal method for stepping forward to the next frame is to use one of these calls to locate the time of the next visual sample. This works well for most media types, including video and text. Unfortunately, it does not work well for MPEG, because QuickTime stores an entire MPEG stream as a single sample. Therefore, stepping to the next sample would actually skip to the end of the entire sequence. To solve this problem, set this flag, which is specifically intended for stepping through frames.

`nextTimeEdgeOk`

Instructs the Movie Toolbox that you are willing to receive information about elements that begin or end at the time specified by the `mediaTimeIn` parameter. Set this flag to 1 to accept this information. This flag is especially useful at the beginning or end of a media.

Discussion

This function allows a derived media handler to return the next step time from the specified media time. The mechanism in QuickTime used for stepping backwards and forwards a frame at a time are the interesting time calls:

`GetMovieNextInterestingTime` (I-480), `GetTrackNextInterestingTime` (I-537), and `GetMediaNextInterestingTime` (I-437).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V-2861)

MediaGetOffscreenBufferSize

MediaGetOffscreenBufferSize

Determines the dimensions of the offscreen buffer.

```
ComponentResult MediaGetOffscreenBufferSize (  
    MediaHandler    mh,  
    Rect            *bounds,  
    short           depth,  
    CTabHandle      ctab );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

bounds

A `Rect` (IV-2399) structure that defines the boundaries of your offscreen buffer.

depth

The depth of the offscreen.

ctab

A handle to the `ColorTable` (IV-2210) structure associated with the offscreen buffer.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

Before the base media handler allocates an offscreen buffer for your derived media handler, it calls this function. The depth and color table used for the buffer are also passed. When this function is called, the `bounds` parameter specifies the size that the base media handler intends to use for your offscreen buffer. You can modify this as appropriate before returning. This capability is useful if your media handler can draw only at particular sizes. It is also useful for implementing antialiased drawing; you can request a buffer that is larger than your destination area and have the base media handler scale the image down for you.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V-2861)

MediaGetPublicInfo

Undocumented

```
ComponentResult MediaGetPublicInfo (
    MediaHandler    mh,
    OSType          infoSelector,
    void            *infoDataPtr,
    Size            *ioDataSize );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from GetMediaHandler (I-432).

infoSelector

Undocumented

infoDataPtr

Undocumented

ioDataSize

Undocumented

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: MediaHandlers.h

MediaGetSampleDataPointer

Allows a derived media handler to obtain a pointer to the sample data for a particular sample number, the size of that sample, and the index of the sample description associated with that sample.

```
ComponentResult MediaGetSampleDataPointer (
    MediaHandler    mh,
    long            sampleNum,
    Ptr             *dataPtr,
    long            *dataSize,
    long            *sampleDescIndex );
```

MediaGetSoundBalance

`mh`

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

`sampleNum`

The number of the sample that is to be loaded.

`dataPtr`

A pointer to a pointer to receive the address of the loaded sample data.

`dataSize`

A pointer to a field that is to receive the size, in bytes, of the sample.

`sampleDescIndex`

A pointer to a long integer. This function returns an index value to the sample description that corresponds to the returned sample data. If you do not want this information, set the parameter to `NIL`.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This function returns a pointer to the data for a particular sample number from a movie data file. It provides access to the base media handler's caching services for sample data. It is a service provided by the base media handler for its clients.

Special Considerations

Each call to this function must be balanced by a call to `MediaReleaseSampleDataPointer` (II-886) or the memory will not be released. This function generally provides better overall performance than `GetMediaSample` (I-443).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V-2861)

MediaGetSoundBalance

Obtains the sound balance of the media handler.

```
ComponentResult MediaGetSoundBalance (
    MediaHandler    mh,
    short           *balance );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

balance

A pointer to an integer. The Movie Toolbox returns the current balance setting of the media handler as a 16-bit, fixed-point value. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Valid balance values range from –1.0 to 1.0. Negative values emphasize the left sound channel, and positive values emphasize the right sound channel; a value of 0 specifies neutral balance.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “Sound Media Handler Functions” (V–2755)

Related Java Methods

```
quicktime.std.movies.media.AudioMediaHandler.getBalance(),
quicktime.std.movies.media.SoundMediaHandler.getBalance(),
quicktime.std.movies.media.MPEGMediaHandler.getBalance()
```

See Also

For the corresponding set function, see `MediaSetSoundBalance` (II–904).

MediaGetSoundBassAndTreble

Gets the bass and treble settings for a media handler.

```
ComponentResult MediaGetSoundBassAndTreble (
    MediaHandler    mh,
    short           *bass,
    short           *treble );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

bass

Undocumented

treble

Undocumented

MediaGetSoundEqualizerBandLevels

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: MediaHandlers.h

See Also

For the corresponding set function, see MediaSetSoundBassAndTreble (II–905).

MediaGetSoundEqualizerBandLevels

Gets the sound equalizer **band** levels for a media handler.

```
ComponentResult MediaGetSoundEqualizerBandLevels (
    MediaHandler    mh,
    UInt8           *bandLevels );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

bandLevels

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: MediaHandlers.h

MediaGetSoundEqualizerBands

Gets the sound equalizer settings for a media handler.

```
ComponentResult MediaGetSoundEqualizerBands (
    MediaHandler    mh,
    MediaEQSpectrumBandsRecordPtr spectrumInfo );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

spectrumInfo

A pointer to a MediaEQSpectrumBandsRecord (IV–2304) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: MediaHandlers.h

See Also

For the corresponding set function, see MediaSetSoundEqualizerBands (II–906).

MediaGetSoundLevelMeterInfo

Gets the right and left sound level meter values for a media handler.

```
ComponentResult MediaGetSoundLevelMeterInfo (  
    MediaHandler      mh,  
    LevelMeterInfoPtr levelInfo );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

levelInfo

A pointer to a LevelMeterInfo (IV–2301) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: MediaHandlers.h

MediaGetSoundLevelMeteringEnabled

Determines if a media handler’s sound level metering capability is enabled.

```
ComponentResult MediaGetSoundLevelMeteringEnabled (  
    MediaHandler      mh,  
    Boolean           *enabled );
```

MediaGetSoundOutputComponent

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

enabled

A pointer to a Boolean; it is `TRUE` if sound level metering is enabled, `FALSE` otherwise.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `MediaHandlers.h`

See Also

For the corresponding set function, see `MediaSetSoundLevelMeteringEnabled` (II–906).

MediaGetSoundOutputComponent

Gets the sound output component associated with a media handler.

```
ComponentResult MediaGetSoundOutputComponent (  
    MediaHandler    mh,  
    Component       *outputComponent );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

outputComponent

An instance of a sound output component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

See Also

For the corresponding set function, see `MediaSetSoundOutputComponent` (II–908).

MediaGetSrcRgn

Specifies an irregular destination display region to the Movie Toolbox.

```
ComponentResult MediaGetSrcRgn (
    MediaHandler    mh,
    RgnHandle       rgn,
    TimeValue       atMediaTime );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

rgn

A handle to a `MacRegion` (IV-2303) structure. When the Movie Toolbox calls your function, this region is initialized to the track's boundary rectangle, which is defined by the width and height fields in the `GetMovieCompleteParams` (IV-2268) structure that you obtain when the Movie Toolbox calls your `MediaInitialize` (II-877) function. Your media handler may then alter this region as appropriate, so that it corresponds to the boundaries of your media's display image. Note that this region is in the track's coordinate system, not the movie's. Do not dispose of this region; it is owned by the Movie Toolbox.

atMediaTime

The time value at which the Movie Toolbox wants to know what the source region is.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if your media does not completely fill the track rectangle during playback. The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (II-894) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: "Managing Graphics Data" (V-2864)

MediaGetTrackOpaque

Determines whether a media is transparent or opaque when displayed.

MediaGetURLLink

```
ComponentResult MediaGetTrackOpaque (
    MediaHandler    mh,
    Boolean         *trackIsOpaque );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

trackIsOpaque

A pointer to a Boolean value. Your media handler must set this value to `TRUE` if your media is semitransparent (that is, you draw in blend mode); otherwise, leave the flag unchanged.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if your media is semitransparent when displayed or if you handle display transfer modes. The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` or `handlerCanTransferMode` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (II–894) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “Managing Graphics Data” (V–2864)

Related Java Methods

`quicktime.std.movies.media.VisualMediaHandler.getTrackOpaque()`

MediaGetURLLink

Undocumented

```
ComponentResult MediaGetURLLink (
    MediaHandler    mh,
    Point           displayWhere,
    Handle          *urlLink );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

displayWhere

Undocumented

urlLink

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

MediaGetUserPreferredCodecs

Retrieves the list of components last passed to the media handler by a call to MediaSetUserPreferredCodecs (II–909).

```
ComponentResult MediaGetUserPreferredCodecs (  
    MediaHandler          mh,  
    CodecComponentHandle *userPreferredCodecs );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from GetMediaHandler (I–432).

userPreferredCodecs

A pointer to a handle containing component identifiers. If the media handler currently has a preferred component list, it will copy that list into a new handle and store the new handle in this variable. If the media handler does not currently have a preferred component list, it will store NIL in this variable. The caller must dispose of this handle.

function result See “Error Codes” (IV–2663). Returns badComponentSelector if the media handler component does not support this call. Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: MediaHandlers.h

See Also

For the corresponding set function, see MediaSetUserPreferredCodecs (II–909).

MediaGetVideoParam

Retrieves the value of the brightness, contrast, hue, sharpness, saturation, black level, or white level of a video image.

```
ComponentResult MediaGetVideoParam (  
    MediaHandler      mh,  
    long              whichParam,  
    unsigned short    *value );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

whichParam

A constant (see below) that specifies the video parameter whose value you want to retrieve.

value

The actual value of the requested video parameter. The meaning of the values vary depending on the implementation.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

whichParam Constants

`kMediaVideoParamBrightness`

The brightness value controls the overall brightness of the digitized video image. Brightness values range from 0 to 65,535, where 0 is the darkest possible setting and 65,535 is the lightest possible setting.

`kMediaVideoParamContrast`

The contrast value ranges from 0 to 65,535, where 0 represents no change to the basic image and larger values increase the contrast of the video image (that is, increase the slope of the transform).

`kMediaVideoParamHue`

Hue is similar to the tint control on a television. It is specified in degrees with complementary colors set 180 degrees apart (red is 0 degrees, green is +120 degrees, and blue is -120 degrees). QuickTime supports hue values that range from 0 (-180 degrees shift in hue) to 65,535 (+179 degrees shift in hue), where 32,767 represents a 0 degrees shift in hue.

`kMediaVideoParamSharpness`

The sharpness value ranges from 0 to 65,535, where 0 represents no sharpness filtering and 65,535 represents full sharpness filtering. Higher values result in a visual impression of increased picture sharpness

kMediaVideoParamSaturation

The saturation value controls color intensity. For example, at high saturation levels, red appears to be red; at low saturation, red appears pink. Valid saturation values range from 0 to 65,535, where 0 is the minimum saturation value and 65,535 specifies maximum saturation.

kMediaVideoParamBlackLevel

Black level refers to the degree of blackness in an image. The highest setting produces an all-black image; on the other hand, the lowest setting yields little, if any, black even with black objects in the scene. Black level values range from 0 to 65,535, where 0 represents the maximum black value and 65,535 represents the minimum black value.

kMediaVideoParamWhiteLevel

White level refers to the degree of whiteness in an image. White level values range from 0 to 65,535, where 0 represents the minimum white value and 65,535 represents the maximum white value

Discussion

This function and `MediaSetVideoParam` (II-910) are currently used by the MPEG media handler.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V-2861)

See Also

For the corresponding set function, see `MediaSetVideoParam` (II-910).

MediaGGetStatus

Reports error conditions to the Movie Toolbox.

```
ComponentResult MediaGGetStatus (
    MediaHandler      mh,
    ComponentResult    *statusErr );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

MediaGSetActiveSegment

`statusErr`

A pointer to a component result field. If you have error information that you would like to report to the Movie Toolbox, place an appropriate result code into the field referred to by this pointer. See “Error Codes” (IV–2663).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if you anticipate that you may encounter an error when playing your media. Because these errors may include such conditions as low memory or missing hardware, you should only rarely create a derived media handler that does not support this function. If your media handler does not support this function, the base media handler always sets the returned result code to `noErr`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “Managing Your Media Handler Component” (V–2861)

MediaGSetActiveSegment

Notifies your derived media handlers of the current active segment.

```
ComponentResult MediaGSetActiveSegment (
    MediaHandler    mh,
    TimeValue       activeStart,
    TimeValue       activeDuration );
```

`mh`

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

`activeStart`

The starting time of the active segment to play. This time value is expressed in your movie’s time scale.

`activeDuration`

A time value that specifies the duration of the active segment. This value is expressed in the movie’s time scale.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Using `SetMovieActiveSegment` (III–1609), an application can limit the time segment of the movie that will be used for play back. Derived media handlers are given the values for the active segment when this function is called by the Movie Toolbox. Active segment information is usually only needed by media handlers that perform their own scheduling.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V–2861)

MediaGSetVolume

Specifies changes to the sound volume setting.

```
ComponentResult MediaGSetVolume (
    MediaHandler    mh,
    short           volume );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

volume

The media’s current volume setting. This value is represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer portion; the low-order 8 bits contain the fractional part. Volume values range from –1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. The Movie Toolbox scales your media’s volume in light of the track’s and movie’s volume settings, but it does not take into account the system speaker volume setting. This value is appropriate for use with the Sound Manager.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if it can play sounds.

Version Notes

Introduced in QuickTime 3 or earlier.

MediaHasCharacteristic

Programming Info

C interface file: `MediaHandlers.h`

MediaHasCharacteristic

Called by Movie Toolbox with a specified characteristic to allow tracks to be identified by various attributes.

```
ComponentResult MediaHasCharacteristic (  
    MediaHandler    mh,  
    OSType          characteristic,  
    Boolean         *hasIt );
```

mh

The Movie Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

characteristic

A constant that specifies the attribute of a track. Examples of characteristics that are currently defined are the constants `VisualMediaCharacteristic` and `AudioMediaCharacteristic`.

hasIt

A pointer to a Boolean value that specifies whether the track has the attribute specified in the *characteristic* parameter. Set this value to TRUE if the attribute applies to your media handler; otherwise, set this value to FALSE.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

You should implement this function for any media handler that has characteristics in addition to spatial ones. If you have set the `handlerHasSpatial` capabilities flag, the base media handler automatically handles the `VisualMediaCharacteristic` constant for you.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V-2861)

MediaHitTestForTargetRefCon

Undocumented

```
ComponentResult MediaHitTestForTargetRefCon (
    MediaHandler    mh,
    long            flags,
    Point           loc,
    long            *targetRefCon );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I-432).

flags

Flags (see below) that define the hit.

loc

Undocumented

targetRefCon

Undocumented

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

flags Constants

mHitTestBounds

The point may only be within the targetRefCon bounding box.

mHitTestImage

The point must be within the shape of the targetRefCon image.

mHitTestInvisible

An invisible targetRefCon may be hit tested.

mHitTestIsClick

The hit is a mouse click (for codecs that want mouse events).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

MediaHitTestTargetRefCon

Undocumented

```
ComponentResult MediaHitTestTargetRefCon (
```

MediaIdle

```
MediaHandler    mh,  
long            targetRefCon,  
long            flags,  
Point           loc,  
Boolean         *wasHit );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I-432).

targetRefCon

Undocumented

flags

Flags (see below) that define the hit.

loc

Undocumented

wasHit

A pointer to a Boolean; it is TRUE if there was a hit, FALSE otherwise.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

flags Constants

mHitTestBounds

The point may only be within the `targetRefCon` bounding box.

mHitTestImage

The point must be within the shape of the `targetRefCon` image.

mHitTestInvisible

An invisible `targetRefCon` may be hit tested.

mHitTestIsClick

The hit is a mouse click (for codecs that want mouse events).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

MediaIdle

Provides processing time to a derived media handler during movie playback.

```
ComponentResult MediaIdle (  
    MediaHandler    mh,  
    TimeValue       atMediaTime,
```



```

long          flagsIn,
long          *flagsOut,
const TimeRecord *movieTime );

```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

atMediaTime

The current time, in your media's time base. You can use this value to obtain the appropriate samples and sample descriptions from your media (using the Movie Toolbox's `GetMediaSample` (I-443) function). Your media handler may then work with the sample data and descriptions as appropriate.

flagsIn

Contains flags (see below) that indicate what the Movie Toolbox wants your media handler to do. These flags are applicable only to media handlers that perform their own scheduling. The toolbox may use none, or it may set one or more flag to 1. Your handler should examine the `flagsIn` parameter each time the Movie Toolbox calls its `MediaIdle` function. The flags in this parameter indicate the actions that your handler may perform. In addition, when you return from your `MediaIdle` function, you should report what you did using the `flagsOut` parameter. You tell the base media handler that you perform your own scheduling by setting the `handlerNoScheduler` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (II-894) function.

flagsOut

Flags (see below) that are contained in a pointer to a long integer that your media handler uses to indicate to the Movie Toolbox what the handler did. You must always set the values of these flags appropriately.

movieTime

A pointer to the movie time value corresponding to the `atMediaTime` parameter. Note that this may differ from the current value returned by `GetMovieTime` (I-495).

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

flagsIn Constants

mMustDraw

Indicates that your media handler must play its media at this time. For graphical media, this means that your handler must draw the appropriate media data on the screen. For sound-based media, your handler must play the media's sounds. If this flag is set to 1, the Movie Toolbox has encountered a new sample in your media.

MediaIdle

`mAtEnd`

Indicates that the current time corresponds to the end of the movie.

`mPartialDraw`

Indicates that only a portion of the available drawing area needs to be redrawn. Whenever you set this flag to 1, the Toolbox calls your component's `MediaGetDrawingRgn` function in order to determine the portion of the image that needs to be redrawn. As an example, consider a full-screen animation. Only rarely is the entire image in motion. Typically, only a small portion of the screen image moves. By using partial redrawing, you can significantly improve the playback performance of such a movie.

`mPreflightDraw`

Indicates that your media handler must not play its media at this time. Your handler may examine the media data and prepare to play it, but you should not draw any graphical data or play any sounds. If this flag is set to 1, your handler must not play its data. If these flags are set to 0, then your media handler is free to decide whether to play the media data or not.

flagsOut Constants

`mDidDraw`

Indicates that your media handler played its media's data with the `handlerHasSpatial` flag set, then it drew the data. Any time your media handler plays its media's data, you should set this flag to 1 when you return from your `MediaIdle` function. The Movie Toolbox uses this information when it is displaying a composited movie; that is, a movie whose image is derived by blending several tracks together. If your media's track is obscured by other, semitransparent tracks, the Movie Toolbox must redraw those other tracks whenever your media's image changes.

`mNeedsToDraw`

Indicates that your media handler needs to play its data. Typically, you use this flag when the Movie Toolbox calls your `MediaIdle` function with the `mPreflightDraw` flag in the `flagsIn` field set to 1, and you discover that you have data that must be played at the current time. Set this flag to 1 if your handler needs to play its media's data.

Discussion

From time to time, your derived media handler component may determine that only a portion of the available drawing area needs to be redrawn. You can signal that condition to the base media handler component by setting the `mPartialDraw` flag to 1 in the flags your component returns to the Movie Toolbox from your `MediaIdle` function. You return these flags using the `flagsOut` parameter. Whenever you set this flag to 1, the Movie Toolbox calls your component's `MediaGetDrawingRgn`

(II-849) function in order to determine the portion of the image that needs to be redrawn.

As an example, consider a full-screen animation. Only rarely is the entire image in motion. Typically, only a small portion of the screen image moves. By using partial redrawing, you can significantly improve the playback performance of such a movie.

Your derived media handler should support this function if you need to do work during movie playback. If you set the handlerNoIdle flag to 1 in the flags parameter of MediaSetHandlerCapabilities (II-894), the Movie Toolbox does not call your MediaIdle function. If you encounter an error, save the result code. The Movie Toolbox polls you for status information using MediaGGetStatus (II-869).

Special Considerations

If your media handler changes any of the settings of the movie's graphics port or graphics world, be sure to restore the original settings before you exit. In addition, note that you may be drawing into a black-and-white graphics port. Finally, be aware that the Movie Toolbox also uses this function to obtain data for QuickDraw pictures. Therefore, if your media handler does not use QuickDraw when drawing to the screen, be sure to examine the picSave field in the graphics port so that you can detect when the Movie Toolbox wants to save an image. Your media handler is then responsible for performing the appropriate display processing.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

Programming summary: "Managing Your Media Handler Component" (V-2861)

See Also

For more information, see also MediaInvalidateRegion (II-879) and MediaGetDrawingRgn (II-849).

MediaInitialize

Prepares a derived media handler component to provide access to its media.

```
ComponentResult MediaInitialize (
    MediaHandler      mh,
    GetMovieCompleteParams *gmc );
```

MediaInitialize

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

gmc

A pointer to a `GetMovieCompleteParams` (IV-2268) structure. You can obtain information about the current media from this structure. You should copy any values you need to save into your derived media handler's local data area. Because this data structure is owned by the Movie Toolbox, you do not need to worry about disposing of any of the data in it.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error. If you return an error, the Movie Toolbox disables the track that uses your media. In cases where your media has just been created, the Movie Toolbox immediately disposes of your media.

Discussion

This function gives your media handler an opportunity to get ready to support the Movie Toolbox. As part of these preparations, your derived media handler should report its capabilities to the base media handler by calling `MediaSetHandlerCapabilities` (II-894). You may choose to examine the data in the `GetMovieCompleteParams` (IV-2268) structure; you may also save values from this structure. If you save references to structures (such as the matte pixel map), do not dispose of the memory associated with these structures. The Movie Toolbox owns these structures.

Note that the Movie Toolbox may call other functions supported by your media handler before it calls your `MediaInitialize` function. In particular, it may call your `MediaGetMediaInfo` (II-853) and `MediaPutMediaInfo` (II-884) functions. However, before the Movie Toolbox tries to do anything with the data in your media, it will call your `MediaInitialize` function. The Movie Toolbox loads the movie's data using functions that are supported by the base media handler; your media handler does not have to support those functions.

Special Considerations

All derived media handlers should support this function. In addition, if your media handler saves values from the `GetMovieCompleteParams` (IV-2268) structure that may change, be sure to support the corresponding functions that allow the Movie Toolbox to report changes to your media handler. For example, if your handler saves the movie time scale from the `movieScale` field, you should also support the `MediaSetMovieTimeScale` (II-899) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “Managing Your Media Handler Component” (V–2861)

MediaInvalidateRegion

Updates the invalidated display region the next time `MediaIdle` is called.

```
ComponentResult MediaInvalidateRegion (  
    MediaHandler    mh,  
    RgnHandle       invalRgn );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

invalRgn

A handle to a region that has been invalidated. Your media handler should not dispose or modify this region. The `invalRgn` parameter is never `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is called by the Movie Toolbox when `UpdateMovie` (III–1981) or `InvalidateMovieRegion` (II–771) is called with a region that intersects your media’s track. Derived media handlers need to implement `MediaInvalidateRegion` only if they can perform efficient updates on a portion of their display area.

If a media handler implements this function, it is responsible for ensuring that the appropriate areas of the screen are updated on the next call to `MediaIdle` (II–874). If a media handler does not implement this function, the base media handler sets the `mMustDraw` flag the next time `MediaIdle` is called.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V–2861)

MediaMakeMediaTimeTable

Called by the base media handler to create a media time table.

MediaMakeMediaTimeTable

```
ComponentResult MediaMakeMediaTimeTable (  
    MediaHandler      mh,  
    long              **offsets,  
    TimeValue         startTime,  
    TimeValue         endTime,  
    TimeValue         timeIncrement,  
    short             firstDataRefIndex,  
    short             lastDataRefIndex,  
    long              *retDataRefSkew );
```

mh

The media handler to create the time table. You can obtain this reference from `GetMediaHandler` (I-432).

offsets

A handle to an unlocked relocatable memory block that is allocated by an application or other software when it calls `QTMovieNeedsTimeTable` (II-1202), `GetMaxLoadedTimeInMovie` (I-423), `MakeTrackTimeTable` (II-793), or `MakeMediaTimeTable` (II-788). Your derived media handler returns the time table for the media in this block. Your media handler has to resize the handle.

startTime

The first point of the media to be included in the time table. This time value is expressed in the media's time coordinate system.

endTime

The last point of the media to be included in the time table. This time value is expressed in the media's time coordinate system.

timeIncrement

The resolution of the time table. The values in a time table are for a points in the media, and these points are separated by the amount of time specified by this parameter. The time value is expressed in the media's time coordinate system.

firstDataRefIndex

The first in the range of data reference indexes you are querying.

lastDataRefIndex

The last in the range of data reference indexes you are querying.

retDataRefSkew

A pointer to the number of entries, i.e., the number of entries in the offset table per data reference.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

The Movie Toolbox calls your derived media handler's `MediaMakeMediaTimeTable` function whenever an application or other software calls the Toolbox's `QTMovieNeedsTimeTable` (II-1202), `GetMaxLoadedTimeInMovie` (I-423), `MakeTrackTimeTable` (II-793), or `MakeMediaTimeTable` (II-788) function. When an application or other software calls one of these functions, it allocates an unlocked relocatable memory block for the time table to be returned and passes a handle to it in the `offsets` parameter. Your derived media handler must resize the block to accommodate the time table it returns.

The time table your derived media handler returns is a two-dimensional array of long integers that is organized so that each row in the table contains values for one data reference. The first column in the table contains values for the time in the media specified by the `startTime` parameter, and each subsequent column contains values for the point in the media that is later by the value specified by the `timeIncrement` parameter. Each long integer value in the table specifies the offset, in bytes, from the beginning of the data reference for that point in the media.

Special Considerations

The number of columns in the table returned by your derived media handler must be equal to $(\text{endTime} - \text{startTime}) / \text{timeIncrement}$, rounded up. It must also return the offset to the next row of the time table, in long integers, in the `retdataRefSkew` parameter. Because of alignment issues, this value is not always equal to $(\text{endTime} - \text{startTime}) / \text{timeIncrement}$ rounded up.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: "Making a Progressive Download Time Table" (V-2866)

MediaMCIsPlayerEvent*Undocumented*

```
ComponentResult MediaMCIsPlayerEvent (
    MediaHandler      mh,
    const EventRecord *e,
    Boolean           *handledIt );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I-432).

MediaPrePrerollBegin

e

A pointer to an EventRecord (IV–2246) structure.

handledIt

A pointer to a Boolean; it is TRUE if the event was handled, FALSE otherwise.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: MediaHandlers.h

MediaPrePrerollBegin

Undocumented

```
ComponentResult MediaPrePrerollBegin (
    MediaHandler      mh,
    TimeValue         time,
    Fixed              rate,
    PrePrerollCompleteUPP completeProc,
    void              *refcon );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

time

Undocumented

rate

Undocumented

completeProc

A PrePrerollCompleteProc (III–2111) callback.

refcon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

MediaPrePrerollCancel

Cancels a media handler pre-preroll operation that was started by `MediaPrePrerollBegin` (II–882).

```
ComponentResult MediaPrePrerollCancel (
    MediaHandler    mh,
    void            *refcon );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

refcon

The reference constant that was passed to your `PrePrerollCompleteProc` (III–2111) callback.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

MediaPreroll

Prepares a media handler for playback.

```
ComponentResult MediaPreroll (
    MediaHandler    mh,
    TimeValue       time,
    Fixed           rate );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

time

The starting time of the media segment to play. This time value is expressed in your media’s time scale.

rate

The rate at which the Movie Toolbox expects to play the media. This is a 32-bit, fixed-point number. Positive values indicate forward rates; negative values correspond to reverse rates.

MediaPutMediaInfo

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

Programming summary: “General Data Management” (V–2861)

MediaPutMediaInfo

Lets a derived media handler store proprietary information in its media.

```
ComponentResult MediaPutMediaInfo (  
    MediaHandler    mh,  
    Handle          h );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from GetMediaHandler (I–432).

h

A handle to storage into which your media handler may place its proprietary information. You determine the format and content of the data that you store in this handle. Your media handler must resize the handle as appropriate before you exit this function. Do not dispose of this handle; it is owned by the Movie Toolbox. The Movie Toolbox uses the base media handler to write this data to your media.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Whenever the Movie Toolbox opens your media, it provides this private data to your media handler by calling your MediaGetMediaInfo (II–853) function. Note that the Movie Toolbox may call this function before it calls your MediaInitialize (II–877) function. Your derived media handler should support this function if you need to store private data in your media.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

Programming summary: “General Data Management” (V–2861)

MediaQueueNonPrimarySourceData

Undocumented

```
ComponentResult MediaQueueNonPrimarySourceData (
    MediaHandler          mh,
    long                  inputIndex,
    long                  dataDescriptionSeed,
    Handle                dataDescription,
    void                  *data,
    long                  dataSize,
    ICMCompletionProcRecordPtr asyncCompletionProc,
    const ICMFrameTimeRecord *frameTime,
    UniversalProcPtr       transferProc,
    void                  *refCon );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I-432).

inputIndex

The ID of the entry in the media's input map to which the queued data corresponds.

dataDescriptionSeed

Undocumented

dataDescription

Undocumented

data

Undocumented

dataSize

Undocumented

asyncCompletionProc

A pointer to an ICMCompletionProcRecord (IV-2279) structure.

frameTime

A pointer to an ICMFrameTimeRecord (IV-2281) structure.

transferProc

Undocumented

refCon

Undocumented

function result See "Error Codes" (IV-2663). Returns noErr if there is no error.

MediaReleaseSampleDataPointer

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

MediaReleaseSampleDataPointer

Balances calls to MediaGetSampleDataPointer (II–859) to release allocated memory.

```
ComponentResult MediaReleaseSampleDataPointer (
    MediaHandler    mh,
    long            sampleNum );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from GetMediaHandler (I–432).

sampleNum

The number of the sample that is to be released.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function should be used only by derived media handlers.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

Programming summary: “General Data Management” (V–2861)

MediaResolveTargetRefCon

Undocumented

```
ComponentResult MediaResolveTargetRefCon (
    MediaHandler    mh,
    QTAtomContainer container,
    QTAtom          atom,
    long            *targetRefCon );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

container

Undocumented

atom

Undocumented

targetRefCon

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: MediaHandlers.h



MediaSampleDescriptionB2N

Undocumented

```
ComponentResult MediaSampleDescriptionB2N (  
    MediaHandler          mh,  
    SampleDescriptionHandle sampleDescriptionH );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

sampleDescriptionH

A handle to a SampleDescription (IV–2414) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

MediaSampleDescriptionChanged

Notifies a media handler that SetMediaSampleDescription (III–1606) has been called for a specified sample description.

```
ComponentResult MediaSampleDescriptionChanged (  
    MediaHandler mh,
```

MediaSampleDescriptionN2B

```
long          index );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

index

The index of the sample description that has been changed.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V–2861)

MediaSampleDescriptionN2B

Undocumented

```
ComponentResult MediaSampleDescriptionN2B (  
    MediaHandler          mh,  
    SampleDescriptionHandle sampleDescriptionH );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

sampleDescriptionH

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

MediaSetActionsCallback

Sets an `ActionsProc` (III–2075) callback for a media handler.

```
ComponentResult MediaSetActionsCallback (
```

```
MediaHandler    mh,
ActionsUPP      actionsCallbackProc,
void            *refcon );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

actionsCallbackProc

A **Universal Procedure Pointer** to an `ActionsProc` (III–2075) callback.

refcon

A pointer to a reference constant to be passed to your callback. Use this constant to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

MediaSetActive

Enables and disables media.

```
ComponentResult MediaSetActive (
    MediaHandler    mh,
    Boolean          enableMedia );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

enableMedia

A Boolean value that indicates whether your media is enabled or disabled. If this parameter is set to `TRUE`, your media is enabled; if the parameter is `FALSE`, your media is disabled.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if you perform your own scheduling or if your media handler uses significant amounts of temporary storage. If you are doing your own scheduling (that is, you have set the `handlerNoScheduler` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (II–894)

MediaSetClip

function), your media handler needs to keep account of the media's active state so that you can properly respond to Movie Toolbox requests. When your media is disabled, you may choose to dispose of temporary storage you have allocated, so that the storage is available to other programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V–2861)

MediaSetClip

Specifies changes to a derived media handler's clipping region.

```
ComponentResult MediaSetClip (  
    MediaHandler    mh,  
    RgnHandle       theClip );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

theClip

A handle to your media's clipping region. Your media handler is responsible for disposing of this region when you are done with it. Note that this region is defined in the movie's coordinate system.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if you draw during playback. The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` and `handlerCanClip` flags to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (II–894) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “Managing Graphics Data” (V–2864)

MediaSetDimensions

Notifies a media handler when its media's spatial dimensions change.

```
ComponentResult MediaSetDimensions (
    MediaHandler    mh,
    Fixed           width,
    Fixed           height );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

width

The width, in pixels, of the track rectangle. This field, along with the *height* field, specifies a rectangle that surrounds the image that is displayed when the current media is played. This value corresponds to the x coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number.

height

The height, in pixels, of the track rectangle. This value corresponds to the y coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

You obtain the initial dimension information from the *width* and *height* fields of the `GetMovieCompleteParams` (IV-2268) structure that the Movie Toolbox provides to your `MediaInitialize` (II-877) function. Your derived media handler should support this function if you draw during playback.

Special Considerations

The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (II-894) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: "Managing Graphics Data" (V-2864)

MediaSetDoMCActionCallback

MediaSetDoMCActionCallback

Sets a DoMCActionProc (III–2082) callback for a media handler.

```
ComponentResult MediaSetDoMCActionCallback (  
    MediaHandler      mh,  
    DoMCActionUPP     doMCActionCallbackProc,  
    void              *refcon );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

doMCActionCallbackProc

A **Universal Procedure Pointer** to a DoMCActionProc (III–2082) callback.

refcon

A pointer to a reference constant to be passed to your callback. Use this constant to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: MediaHandlers.h

MediaSetGraphicsMode

Sets the graphics mode and blend color of any media handler.

```
ComponentResult MediaSetGraphicsMode (  
    MediaHandler      mh,  
    long              mode,  
    const RGBColor    *opColor );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from GetMediaHandler (I–432).

mode

The graphics mode of the media handler; see “Graphics Transfer Modes” (IV–2670).

opColor

A pointer to the color for use in blending and transparent operations. The media handler passes this color to QuickDraw as appropriate when you draw in addPin, subPin, blend, transparent, or graphicsModeStraightAlphaBlend mode.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

Programming summary: “Managing Graphics Data” (V–2864), “Video Media Handler Functions” (V–2755)

Related Java Methods

quicktime.std.movies.media.VisualMediaHandler.setGraphicsMode()

See Also

You can retrieve the graphics mode and blend color currently in use by any media handler by calling MediaGetGraphicsMode (II–852).

MediaSetGWorld

Lets a derived media handler learn about changes to its media’s graphic environment.

```
ComponentResult MediaSetGWorld (
    MediaHandler    mh,
    CGrafPtr        aPort,
    GDHandle        aGD );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from GetMediaHandler (I–432).

aPort

A pointer to the new graphics port. Note that this may be either a color or a black-and-white port.

aGD

A handle to the new graphics device.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

MediaSetHandlerCapabilities

Discussion

Your derived media handler should support this function if you perform specialized graphics processing or if you are using the Image Compression Manager to decompress your media. Note that when the Movie Toolbox calls your `MediaIdle` (II–874) function, it supplies you with information about the current graphics environment. Consequently, you do not need to support the `MediaSetGWorld` function in order to draw during playback. However, if your media data is compressed and you are using the Image Compression Manager to decompress sequences, you may need to provide updated graphics environment information before playback.

You obtain the initial graphics environment information from the `moviePort` and `movieGD` fields of the `GetMovieCompleteParams` (IV–2268) structure that the Movie Toolbox provides to your `MediaInitialize` (II–877) function. The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (II–894) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “Managing Graphics Data” (V–2864)

MediaSetHandlerCapabilities

Lets a derived media handler report its capabilities to the base media handler.

```
ComponentResult MediaSetHandlerCapabilities (
    MediaHandler    mh,
    long            flags,
    long            flagsMask );
```

`mh`

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

`flags`

Flags (see below) that specify the capabilities of your derived media handler. This parameter contains a number of flags, each of which corresponds to a particular feature. You may work with more than one flag at a time. Be sure to set unused flags to 0.

flagsMask

Indicates which flags in the `flags` parameter are to be considered in this operation. For each bit in the `flags` parameter that you want the base media handler to consider, you must set the corresponding bit in the `flagsMask` parameter to 1. Set unused flags to 0. This allows you to work with a single flag without altering the settings of other flags.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants**handlerHasSpatial**

Indicates that your handler does spatial processing. If you set this flag to 1, the Movie Toolbox informs your derived media handler about changes to the graphics environment or spatial representation of your media.

handlerCanClip

Indicates that your media handler can perform clipping. If you set this flag to 1, the Movie Toolbox calls your `MediaSetClip` (II–890) function whenever the clipping region changes.

handlerCanMatte

Reserved for Apple. Do not set this flag to 1.

handlerCanTransferMode

Indicates that you can work with transfer modes other than source copy or **dither** copy. If you set this flag to 1, the Movie Toolbox calls your `MediaGetTrackOpaque` (II–865) function to determine whether your track is transparent.

handlerNeedsBuffer

Indicates that your media handler needs help during playback. If you set this flag to 1, the base media handler allocates an offscreen buffer and handles all display transformations for you.

handlerNoIdle

Indicates that your derived media handler does not need any processing time during playback. If you set this flag to 1, the Movie Toolbox never calls your `MediaIdle` (II–874) function. This is useful for media handlers that store data in a media, but do not play that data.

handlerNoScheduler

Indicates that your media handler performs special processing during playback. Normally, the Movie Toolbox calls your `MediaIdle` (II–874) function only when it is time for your handler to draw data from a new media sample. If you set this flag to 1, the Movie Toolbox calls that function other times as well,

MediaSetHints

so that your media handler can prepare for playback or perform other necessary processing.

`handlerWantsTime`

Indicates that your media handler needs additional processing time. If you set this flag to 1, the Movie Toolbox calls your `MediaIdle` (II–874) function as often as possible.

`handlerCGrafPortOnly`

Indicates that your media handler can only draw into color graphics ports. If you set this flag to 1, the base media handler performs the necessary processing to display your color media on a black-and-white graphics port. This involves drawing to an offscreen buffer and then transferring the image to the screen.

Discussion

Your media handler may call this function at any time. In general, you should call it from your `MediaInitialize` (II–877) function, so that you report your capabilities to the base media handler before the Movie Toolbox starts working with your media. You may call this function again later, in response to changing conditions. For example, if your media handler receives a matrix that it cannot accommodate from the `MediaSetMatrix` (II–897) function, you can allow the base media handler to handle your drawing by calling this function and setting the `handlerNeedsBuffer` flag in both the `flags` parameter and the `flagsMask` parameter to 1.

Special Considerations

Note that this function is provided by the base media handler; your media handler does not support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “Reporting Capabilities to the Base Media Handler” (V–2866)

MediaSetHints

Implements the appropriate behavior for the various media hints such as scrub mode and high-quality mode.

```
ComponentResult MediaSetHints (
    MediaHandler    mh,
    long            hints );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

hints

All hint bits that currently apply to the given media.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

When an application calls `SetMoviePlayHints` (III-1623) or `SetMediaPlayHints` (III-1601), your media handler's `MediaSetHints` routine is called.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: "General Data Management" (V-2861)

MediaSetMatrix

Tells a media handler about changes to either the movie matrix or the track matrix.

```
ComponentResult MediaSetMatrix (
    MediaHandler    mh,
    MatrixRecord    *trackMovieMatrix );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

trackMovieMatrix

A pointer to the matrix that transforms your media's pixels into the movie's coordinate system. The Movie Toolbox obtains this matrix by concatenating the track matrix and the movie matrix. You should use this matrix whenever you are displaying graphical data from your media.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

You obtain the initial matrix from the `trackMovieMatrix` field of the `GetMovieCompleteParams` (IV-2268) structure that the Movie Toolbox provides to your `MediaInitialize` (II-877) function. Your derived media handler should support this function if you draw during playback.

MediaSetMediaTimeScale

Special Considerations

The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (II–894) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “Managing Graphics Data” (V–2864)

MediaSetMediaTimeScale

Notifies a media handler that its media’s time scale has been changed.

```
ComponentResult MediaSetMediaTimeScale (
    MediaHandler    mh,
    TimeScale       newTimeScale );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

newTimeScale

Specifies your media’s new time scale.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You obtain the initial media time scale information from the `mediaScale` field of the `GetMovieCompleteParams` (IV–2268) structure that the Movie Toolbox provides to your `MediaInitialize` (II–877) function.

Special Considerations

Your derived media handler should support this function if your media handler stores time information that pertains to its media.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V–2861)

MediaSetMovieTimeScale

Notifies a media handler that the movie's time scale has been changed.

```
ComponentResult MediaSetMovieTimeScale (
    MediaHandler    mh,
    TimeScale       newTimeScale );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

newTimeScale

The movie's new time scale.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

You obtain the initial movie time scale information from the `movieScale` field of the `GetMovieCompleteParams` (IV-2268) structure that the Movie Toolbox provides to your `MediaInitialize` (II-877) function.

Special Considerations

Your derived media handler should support this function if your media handler stores time information in the movie's time coordinate system.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: "General Data Management" (V-2861)

MediaSetNonPrimarySourceData

Allows a media handler to support receiving media data from other media handlers.

```
ComponentResult MediaSetNonPrimarySourceData (
    MediaHandler    mh,
    long            inputIndex,
    long            dataDescriptionSeed,
    Handle          dataDescription,
    void            *data,
    long            dataSize,
    ICMCompletionProcRecordPtr asyncCompletionProc,
```

MediaSetNonPrimarySourceData

```
UniversalProcPtr      transferProc,  
void                  *refCon );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

inputIndex

This value is the ID of the entry in the media's input map to which the data provided by the call corresponds.

dataDescriptionSeed

This value is changed each time the `dataDescription` has changed. This allows for a quick check by the media handler to see if the `dataDescription` has changed.

dataDescription

A handle to a data structure describing the input data.

data

A pointer to the input data. This pointer must contain a 32-bit address.

dataSize

The size of the sample in bytes.

asyncCompletionProc

A pointer to a `ICMCompletionProcRecord` (IV-2279) structure. If `asyncCompletionProc` is set to `NIL`, the data pointer will be valid only for the duration of this call. If `asyncCompletionProc` is not `NIL`, it contains an `ICMCompletionProcRecord` (IV-2279) structure that must be called when your media handler is done with the provided data pointer.

transferProc

A routine that allows the application to transform the type of the input data to the kind of data preferred by the codec. The client of the codec passes the source data in the form most convenient for it. If the codec needs the data in another form, it can negotiate with the caller or directly with the Image Compression Manager to obtain the required data format.

refCon

A reference constant, defined as a void pointer. Your application specifies the value of this reference constant in the function structure you pass to the media handler.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

There are two tasks required to support modifier tracks in a derived media handler: sending and receiving. The base media handler takes care of sending data for all its clients. Therefore, authors of derived media handlers do not usually need to implement sending data support.

Receiving data is a more complex situation. The base media handler takes care of input types that it understands. The base media handler supports the following types of data: If a media handler wants to support receiving other types of data it must implement `MediaSetNonPrimarySourceData`. `MediaSetNonPrimarySourceData` is called by modified tracks to supply the current data for each input. All unrecognized input types should be delegated to the base media handler so that they can be handled.

The following is a basic shell implementation of a derived media handler's `MediaSetNonPrimarySourceData` function. Note that your derived media handler must delegate all input types it does not handle to the base media handler:

```
// MySetNonPrimarySourceData coding example
kTrackModifierTypeMatrix
kTrackModifierTypeGraphicsMode
kTrackModifierTypeClip
kTrackModifierTypeVolume
kTrackModifierTypeBalance

pascal ComponentResult MySetNonPrimarySourceData( MyGlobals store,
    long inputIndex, long dataDescriptionSeed, Handle dataDescription,
    void *data, long dataSize,
    ICMCompletionProcRecordPtr asyncCompletionProc,
    UniversalProcPtr transferProc, void *refCon )
{
    ComponentResult err = noErr;
    QTAtom inputAtom;
    QTAtom typesAtom;
    long inputType;

    // determine what kind of input this is
    inputAtom = QTFindChildByID(store->inputMap,
        kParentAtomIsContainer, kTrackModifierInput, inputIndex, NIL);
    if (!inputAtom) {
        err = cannotFindAtomErr;
    }
}
```

MediaSetPublicInfo

```
        goto bail;
    }
    typesAtom = QTFindChildByID(store->inputMap, inputAtom,
        kTrackModifierType, 1, NIL);
    err = QTCopyAtomDataToPtr(store->inputMap, typesAtom, FALSE,
        sizeof(inputType), &inputType, NIL);
    if (err) goto bail;

    switch(inputType) {
        case kMyInputType:
            if (data == NIL) {
                // no data, reset to default value
            }
            else {
                // use this data
                // when done, notify caller we're done with this data
                if (asyncCompletionProc)
                    CallICMCompletionProc(
                        asyncCompletionProc->completionProc,
                        noErr, codecCompletionSource | codecCompletionDest,
                        asyncCompletionProc->completionRefCon);
            }
            break;
    }
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

Programming summary: “General Data Management” (V–2861)

MediaSetPublicInfo

Undocumented

```
ComponentResult MediaSetPublicInfo (
    MediaHandler    mh,
    OSType          infoSelector,
    void            *infoDataPtr,
    Size            dataSize );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

infoSelector

Undocumented

infoDataPtr

Undocumented

dataSize

Undocumented

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `MediaHandlers.h`

MediaSetRate

Sets a media's playback rate.

```
ComponentResult MediaSetRate (
    MediaHandler    mh,
    Fixed           rate );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

rate

A 32-bit, fixed-point number that indicates your media's new effective playback rate. This effective rate accounts for any master time bases that may be in use with the current movie. Positive values represent forward rates and negative values indicate reverse rates.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

You obtain the initial rate information from the `effectiveRate` field of the `GetMovieCompleteParams` (IV-2268) structure that the Movie Toolbox provides to your `MediaInitialize` (II-877) function. Your derived media handler should support this function if you perform your own scheduling; that is, you have set the

MediaSetScreenLock

handlerNoScheduler flag to 1 in the flags parameter of MediaSetHandlerCapabilities (II-894). Your media handler can use this function to determine when your media is playing, and the direction and rate of playback. This information can help you prepare for playback more efficiently.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

Programming summary: “General Data Management” (V-2861)

MediaSetScreenLock

Locks the display screen for a media handler.

```
ComponentResult MediaSetScreenLock (
    MediaHandler    mh,
    Boolean         lockIt );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I-432).

lockIt

Pass TRUE to lock the screen, FALSE to unlock it.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: MediaHandlers.h

MediaSetSoundBalance

Sets the sound balance of the media handler.

```
ComponentResult MediaSetSoundBalance (
    MediaHandler    mh,
    short           balance );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I-432).

balance

The balance setting of the media handler as a 16-bit, fixed-point value. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Valid balance values range from –1.0 to 1.0.

Negative values emphasize the left sound channel, and positive values emphasize the right sound channel; a value of 0 specifies neutral balance.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

Programming summary: “Sound Media Handler Functions” (V–2755)

Related Java Methods

quicktime.std.movies.media.AudioMediaHandler.setBalance(),

quicktime.std.movies.media.SoundMediaHandler.setBalance(),

quicktime.std.movies.media.MPEGMediaHandler.setBalance()

See Also

For the corresponding get function, see MediaGetSoundBalance (II–860).

MediaSetSoundBassAndTreble

Sets the bass and treble controls for a media handler.

```
ComponentResult MediaSetSoundBassAndTreble (
    MediaHandler    mh,
    short           bass,
    short           treble );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

bass

Undocumented

treble

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

MediaSetSoundEqualizerBands

Programming Info

C interface file: `MediaHandlers.h`

See Also

For the corresponding get function, see `MediaGetSoundBassAndTreble` (II–861).

MediaSetSoundEqualizerBands

Sets sound equalizer **bands** for a media handler.

```
ComponentResult MediaSetSoundEqualizerBands (  
    MediaHandler      mh,  
    MediaEQSpectrumBandsRecordPtr  spectrumInfo );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

spectrumInfo

A pointer to a `MediaEQSpectrumBandsRecord` (IV–2304) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `MediaHandlers.h`

See Also

For the corresponding get function, see `MediaGetSoundEqualizerBands` (II–862).

MediaSetSoundLevelMeteringEnabled

Enables or disables sound level metering for a media handler.

```
ComponentResult MediaSetSoundLevelMeteringEnabled (  
    MediaHandler      mh,  
    Boolean            enable );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

enable

Pass `TRUE` to enable sound level metering, `FALSE` to disable it.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `MediaHandlers.h`

See Also

For the corresponding get function, see `MediaGetSoundLevelMeteringEnabled` (II–863).

MediaSetSoundLocalizationData

Supports 3D sound capabilities in a media handler that plays sound.

```
ComponentResult MediaSetSoundLocalizationData (
    MediaHandler    mh,
    Handle          data );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

data

The data passed to your media handler, in the format of a Macintosh Sound Sprockets `SSpLocalizationData` (IV–2460) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This routine is passed a handle containing the new `SSpLocalizationData` (IV–2460) structure to use. If the handle is `NIL`, it indicates that no 3D sound effects should be used. If you implement this routine, and return `noErr` as the result, it is assumed that your media handle assumes responsibility for disposing of the data handle passed. If the implementation of this routine returns an error, the caller will dispose of the handle. The reason for this behavior is to minimize the copying of the settings handle, making it easier for developers to implement this function. This function is called regardless of whether the 3D sound settings were set on the track using `SetTrackSoundLocalizationSettings` (III–1666) or via a modifier track mechanism.

Special Considerations

If you are creating a media handler that plays sound and wish to support 3D sound capabilities, you need to implement this routine.

MediaSetSoundOutputComponent

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “Managing Sound Localization” (V–2866)

MediaSetSoundOutputComponent

Sets the sound output component for a media handler.

```
ComponentResult MediaSetSoundOutputComponent (
    MediaHandler      mh,
    Component          outputComponent );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

outputComponent

An instance of a sound output component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

See Also

For the corresponding get function, see `MediaGetSoundOutputComponent` (II–864).

MediaSetTrackInputMapReference

Provides a derived media handler with an updated input map.

```
ComponentResult MediaSetTrackInputMapReference (
    MediaHandler      mh,
    QTAtomContainer    inputMap );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I–432).

inputMap

The media input map for this operation. Do not modify or dispose of the input map provided.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

When an application modifies the media input map, this function provides the derived media handler with the updated input map. When this function is called, the media handler should store the updated input map and recheck the types of all inputs, if it is caching this information. The input map reference passed to this function should not be disposed of or modified by the media handler.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

Programming summary: “General Data Management” (V–2861)

MediaSetUserPreferredCodecs

Requests that a media handler favor specified codec components when selecting components with which to play media.

```
ComponentResult MediaSetUserPreferredCodecs (
    MediaHandler          mh,
    CodecComponentHandle   userPreferredCodecs );
```

mh

The Toolbox’s connection to your derived media handler. You can obtain this reference from GetMediaHandler (I–432).

userPreferredCodecs

A handle containing component identifiers. The media handler component will make its own copy of this handle. The components should be specified in order from most preferred to least preferred. Pass NIL to invalidate the standing request without substituting another.

function result See “Error Codes” (IV–2663). Returns badComponentSelector if the media handler component does not support this call. Returns noErr if there is no error.

MediaSetVideoParam

Discussion

This method does not guarantee that the specified components will be used; other factors may take precedence. Components that are preferred may not be used if they can't be part of the chain required to play the media; for example, if they don't handle the pixel format or the video output.

Here is an example of code that uses this function:

```
CodecComponentHandle userPreferredCodecs = nil;
ComponentDescription cd = { decompressorComponentType,
                            myPreferredCodecType,
                            myPreferredCodecManufacturer,
                            0,
                            0 };

CodecComponent      c = FindNextComponent( 0, &cd );
MediaHandler        myMedia;
OSErr               err;

PtrToHand( &c, (Handle*)&userPreferredCodecs, sizeof(c) );

myMedia = GetMediaHandler( GetTrackMedia( track ) );

err = MediaSetUserPreferredCodecs( myMedia, userPreferredCodecs );
DisposeHandle( (Handle)userPreferredCodecs );
```

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: MediaHandlers.h

See Also

For the corresponding get function, see MediaGetUserPreferredCodecs (II-867).

MediaSetVideoParam

Lets you dynamically adjust the brightness, contrast, hue, sharpness, saturation, black level, and white level of a video image.

```
ComponentResult MediaSetVideoParam (
    MediaHandler      mh,
    long               whichParam,
    unsigned short     *value );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

whichParam

A constant (see below) that specifies the video parameter you want to adjust.

value

The actual value of the video parameter. The meaning of the values vary depending on the implementation.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

whichParam Constants

`kMediaVideoParamBrightness`

The brightness value controls the overall brightness of the digitized video image. Brightness values range from 0 to 65,535, where 0 is the darkest possible setting and 65,535 is the lightest possible setting.

`kMediaVideoParamContrast`

The contrast value ranges from 0 to 65,535, where 0 represents no change to the basic image and larger values increase the contrast of the video image (that is, increase the slope of the transform).

`kMediaVideoParamHue`

Hue is similar to the tint control on a television. It is specified in degrees with complementary colors set 180 degrees apart (red is 0 degrees, green is +120 degrees, and blue is -120 degrees). QuickTime supports hue values that range from 0 (-180 degrees shift in hue) to 65,535 (+179 degrees shift in hue), where 32,767 represents a 0 degrees shift in hue.

`kMediaVideoParamSharpness`

The sharpness value ranges from 0 to 65,535, where 0 represents no sharpness filtering and 65,535 represents full sharpness filtering. Higher values result in a visual impression of increased picture sharpness.

`kMediaVideoParamSaturation`

The saturation value controls color intensity. For example, at high saturation levels, red appears to be red; at low saturation, red appears pink. Valid saturation values range from 0 to 65,535, where 0 is the minimum saturation value and 65,535 specifies maximum saturation.

`kMediaVideoParamBlackLevel`

Black level refers to the degree of blackness in an image. The highest setting produces an all-black image; on the other hand, the lowest setting yields little, if any, black even with black objects in the scene. Black level values range from



MediaTargetRefConsEqual

0 to 65,535, where 0 represents the maximum black value and 65,535 represents the minimum black value.

kMediaVideoParamWhiteLevel

White level refers to the degree of whiteness in an image. White level values range from 0 to 65,535, where 0 represents the minimum white value and 65,535 represents the maximum white value.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: “General Data Management” (V–2861)

See Also

For the corresponding get function, see `MediaGetVideoParam` (II–868). The `MediaSetVideoParam` and `MediaGetVideoParam` (II–868) functions are currently used by the MPEG media handler.

MediaTargetRefConsEqual

Undocumented

```
ComponentResult MediaTargetRefConsEqual (
    MediaHandler    mh,
    long            firstRefCon,
    long            secondRefCon,
    Boolean         *equal );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

firstRefCon

Undocumented

secondRefCon

Undocumented

equal

A pointer to a Boolean; it is TRUE if *firstRefCon* and *secondRefCon* are equal, FALSE otherwise.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: MediaHandlers.h

MediaTimeBaseChanged

Undocumented

```
ComponentResult MediaTimeBaseChanged (
    MediaHandler    mh );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

function result *Undocumented*

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: MediaHandlers.h

MediaTimeToSampleNum

Lets you find the sample that contains the data for a specified time.

```
void MediaTimeToSampleNum (
    Media        theMedia,
    TimeValue    time,
    long         *sampleNum,
    TimeValue    *sampleTime,
    TimeValue    *sampleDuration );
```

theMedia

The media for this operation. You obtain this media identifier from such functions as NewTrackMedia (II–1120) and GetTrackMedia (I–535).

time

The time for which you are retrieving sample information. You must specify this value in the media’s time scale.

sampleNum

A pointer to a long integer that is to receive the sample number. The Movie Toolbox returns the sample number that identifies the sample that contains data for the time specified by the time parameter.

MediaTrackEdited

`sampleTime`

A pointer to a time value. The `MediaTimeToSampleNum` function updates this time value to indicate the starting time of the sample that contains data for the time specified by the `time` parameter. This time value is expressed in the media's time scale. Set this parameter to `NIL` if you don't want this information.

`sampleDuration`

A pointer to a time value. The Movie Toolbox returns the duration of the sample that contains data for the time specified by the `time` parameter. This time value is expressed in the media's time scale. Set this parameter to `NIL` if you don't want this information.

function result You can access this function's error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Samples” (V-2742)

Related Java Methods

`quicktime.std.movies.media.Media.timeToSampleNum()`

See Also

`MediaTimeToSampleNum` does not account for edits applied to the media by a movie's tracks. If you want to work with edits, use the functions that allow you to look for interesting times, such as `GetMediaNextInterestingTime` (I-437), `GetMovieNextInterestingTime` (I-480), and `GetTrackNextInterestingTime` (I-537).

MediaTrackEdited

Informs a derived media handler about edits to its track.

```
ComponentResult MediaTrackEdited (
    MediaHandler    mh );
```

`mh`

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if you are caching location information about track edits, or if you are using any time values in the movie's time base. Whenever the Movie Toolbox calls this function, your media handler should recalculate this type of information.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: "General Data Management" (V-2861)

MediaTrackPropertyAtomChanged

Notifies the derived media handler whenever its media property atom has changed.

```
ComponentResult MediaTrackPropertyAtomChanged (  
    MediaHandler    mh );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

`MediaTrackPropertyAtomChanged` is called whenever `SetMediaPropertyAtom` (III-1604) is called. If the media handler uses information from the property atom, it should rebuild the information at this time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: "General Data Management" (V-2861)

MediaTrackReferencesChanged

Notifies the derived media handler whenever the track references in the movie change.

```
ComponentResult MediaTrackReferencesChanged (
```

MediaVideoOutputChanged

```
MediaHandler mh );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

When an application creates, modifies, or deletes a track reference, the media handler's `MediaTrackReferencesChanged` function is called. When this function is called, a media handler should rebuild all information about track references and reset its values for all media inputs to their default values.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `MediaHandlers.h`

Programming summary: "General Data Management" (V-2861)

MediaVideoOutputChanged

Undocumented

```
ComponentResult MediaVideoOutputChanged (
    MediaHandler mh,
    ComponentInstance vout );
```

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (I-432).

vout

Undocumented

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `MediaHandlers.h`

MIDIImportGetSettings

Obtains settings that control the importation of MIDI files.

```
ComponentResult MIDIImportGetSettings (
    TextExportComponent    ci,
    long                   *setting );
```

ci

A text export component instance used to import a MIDI file. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

setting

Flags (see below) that control the importation of MIDI files. The flags correspond to the checkboxes in the MIDI Import Options dialog box.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

setting Constants

`kMIDIImportSilenceBefore`

Adds one second of silence before the first note.

`kMIDIImportSilenceAfter`

Adds one second of silence after the last note.

`kMIDIImport20Playable`

Imports only MIDI data that can be used with QuickTime 2.0. The imported data does not include program changes and has at most 32 parts.

`kMIDIImportWantLyrics`

Imports karaoke lyrics as a text track.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Importing MIDI Files” (V–2924)

See Also

For the corresponding set function, see `MIDIImportSetSettings` (II–917).

MIDIImportSetSettings

Define settings that control the importation of MIDI files.

MovieExecuteWiredActions

```
ComponentResult MIDIImportSetSettings (
    TextExportComponent    ci,
    long                    setting );
```

ci

A text export component instance used to import a MIDI file. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

setting

Flags (see below) that control the importation of MIDI files. The flags correspond to the checkboxes in the MIDI Import Options dialog box.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

setting Constants

`kMIDIImportSilenceBefore`

Adds one second of silence before the first note.

`kMIDIImportSilenceAfter`

Adds one second of silence after the last note.

`kMIDIImport20Playable`

Imports only MIDI data that can be used with QuickTime 2.0. The imported data does not include program changes and has at most 32 parts.

`kMIDIImportWantLyrics`

Imports karaoke lyrics as a text track.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Importing MIDI Files” (V–2924)

See Also

For the corresponding get function, see `MIDIImportGetSettings` (II–917).

MovieExecuteWiredActions

Undocumented

```
OSErr MovieExecuteWiredActions (
    Movie            theMovie,
    long             flags,
    QTAtomContainer  actions );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as NewMovie (II–1069), NewMovieFromFile (II–1080), and NewMovieFromHandle (II–1084).

flags

Undocumented

actions

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

flags Constant

movieExecuteWiredActionDontExecute

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

MovieExportAddDataSource

Defines a data source for use with an export operation performed by MovieExportFromProceduresToDataRef (II–922).

```
ComponentResult MovieExportAddDataSource (
    MovieExportComponent    ci,
    OSType                  trackType,
    TimeScale                scale,
    long                    *trackID,
    MovieExportGetPropertyUPP getPropertyProc,
    MovieExportGetDataUPP   getDataProc,
    void                    *refCon );
```

ci

A movie export component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

trackType

The type of media provided by this data source. This normally corresponds to a QuickTime media type such as VideoMediaType or SoundMediaType.

MovieExportDisposeGetDataAndPropertiesProcs

scale

The time scale for time values passed to `getDataProc` parameter. If the source data is being taken from a QuickTime track, this value is typically the media's time scale.

trackID

An identifier for the data source. This identifier is returned from the call.

getPropertyProc

A `MovieExportGetPropertyProc` (III–2102) callback that provides information about processing source samples.

getDataProc

A `MovieExportGetDataProc` (III–2101) callback the export component uses to request sample data.

refCon

Passed to the procedures specified in the `getPropertyProc` and `getDataProc` parameters. Use this parameter to point to a data structure containing any information your callbacks need.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Before starting an export operation, all the data sources must be defined by calling this function once for each data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Movie Data” (V–2858)

MovieExportDisposeGetDataAndPropertiesProcs

Disposes of the memory associated with the procedures returned by `MovieExportNewGetDataAndPropertiesProcs` (II–927).

```
ComponentResult MovieExportDisposeGetDataAndPropertiesProcs (
    MovieExportComponent      ci,
    MovieExportGetPropertyUPP  getPropertyProc,
    MovieExportGetDataUPP     getDataProc,
    void                      *refCon );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

getPropertyProc

A `MovieExportGetPropertyProc` (III–2102) callback that provides information about processing source samples.

getDataProc

A `MovieExportGetDataProc` (III–2101) callback that the export component uses to request sample data.

refCon

Passed to the procedures specified in the `getPropertyProc` and `getDataProc` parameters. Use this parameter to point to a data structure containing any information your callbacks need.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Movie Data” (V–2858)

MovieExportDoUserDialog

Requests that a component display its user dialog box.

```
ComponentResult MovieExportDoUserDialog (
    MovieExportComponent    ci,
    Movie                   theMovie,
    Track                   onlyThisTrack,
    TimeValue               startTime,
    TimeValue               duration,
    Boolean                 *canceled );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

theMovie

The movie containing the data to be exported.

MovieExportFromProceduresToDataRef

`onlyThisTrack`

Specifies that the export component should only attempt to export the data from a single track. If this parameter is set to `NIL`, the exporter should attempt to export the entire movie, or all of the tracks in the movie that it can export. For example, an audio export component might export multiple audio tracks, mixing them if necessary. If this parameter is not `NIL`, the exporter should attempt to export only the specified track.

`startTime`

The movie time at which to begin the export operation. If you pass 0, the operation should start at the beginning of the movie or track.

`duration`

The duration, in movie timescale units, of the segment to be exported. To export the entire movie, or an entire track, pass in the value returned by `GetMovieDuration` (I–470) or `GetTrackDuration` (I–529), minus the value passed in `startTime`, as described above.

`canceled`

A pointer to a Boolean value. Your component should set this value to `TRUE` if the user cancels the dialog box, otherwise `FALSE`. If the user cancels the dialog box, your component should revert to its settings as they were before executing this function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Export Components” (V–2859)

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.doUserDialog()`

MovieExportFromProceduresToDataRef

Exports data provided by `MovieExportAddDataSource` (II–919) to a specified location.

```
ComponentResult MovieExportFromProceduresToDataRef (
    MovieExportComponent    ci,
    Handle                  dataRef,
    OSType                  dataRefType );
```


ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

dataRef

The data reference for the export operation.

dataRefType

The type identifier for the data reference specified by *dataRef*.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function exports data provided by `MovieExportAddDataSource` (II–919) to a location specified by *dataRef* and *dataRefType*. Typically *dataRef* contains a Macintosh file **alias** and *dataRefType* is set to `rAliasType`.

Special Considerations

Movie data export components that support export operations from procedures must set the `canMovieExportFromProcedures` flag in their component flags.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Movie Data” (V–2858)

MovieExportGetAuxiliaryData

Retrieves additional data from a component.

```
ComponentResult MovieExportGetAuxiliaryData (
    MovieExportComponent    ci,
    Handle                  dataH,
    OSType                  *handleType );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

dataH

A handle that is to be filled with the additional data. Your component should resize this handle as appropriate. Your component is not responsible for disposing of this handle.

MovieExportGetCreatorType

handleType

A pointer to the type of data you place in the handle specified by the data parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your component should expect the application to call this function after the export process ends.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Export Components” (V–2859), “Exporting Movie Data” (V–2858)

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.getAuxiliaryData()`

MovieExportGetCreatorType

Undocumented

```
ComponentResult MovieExportGetCreatorType (  
    MovieExportComponent    ci,  
    OSType                  *creator );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

creator

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.getCreatorType()`

MovieExportGetFileNameExtension

Undocumented

```
ComponentResult MovieExportGetFileNameExtension (
    MovieExportComponent    ci,
    OSType                  *extension );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

extension

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.getExportFileNameExtension()`

MovieExportGetSettingsAsAtomContainer

Retrieves the current settings from the movie export component.

```
ComponentResult MovieExportGetSettingsAsAtomContainer (
    MovieExportComponent    ci,
    QTAtomContainer         *settings );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

settings

The address where the newly-created atom container should be stored by the call. The caller is responsible for disposing of the returned QT atom container.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

MovieExportGetShortFileTypeString

Discussion

Applications can call this function to obtain a correctly formatted atom container to use with `MovieExportSetSettingsFromAtomContainer` (II–932). This might be done after a call to `MovieExportDoUserDialog` (II–921), for example, to apply the user-obtained settings to a series of exports.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Movie Data” (V–2858)

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.getExportSettingsFromAtomContainer`

`quicktime.std.movies.AtomContainer.fromMovieExporter()`

MovieExportGetShortFileTypeString

Undocumented

```
ComponentResult MovieExportGetShortFileTypeString (
    MovieExportComponent    ci,
    Str255                  typeString );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

typeString

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.getExportShortFileTypeString()`

MovieExportGetSourceMediaType

Returns either the track type if a movie export component is track-specific or 0 if it is track-independent.

```
ComponentResult MovieExportGetSourceMediaType (
    MovieExportComponent    ci,
    OSType                  *mediaType );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

mediaType

The track type if the component is track-specific or 0 if it is track-independent.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This routine returns the same values that were previously stored in the `componentManufacturer` field of the `ComponentDescription` (IV–2212) structure. This frees up the field to be used for the manufacturer.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.getExportSourceMediaType()`

MovieExportNewGetDataAndPropertiesProcs

Returns `MovieExportGetPropertyProc` (III–2102) and `MovieExportGetDataProc` (III–2101) callbacks that can be passed to `MovieExportAddDataSource` (II–919) to create a new data source.

```
ComponentResult MovieExportNewGetDataAndPropertiesProcs (
    MovieExportComponent    ci,
    OSType                  trackType,
    TimeScale                *scale,
    Movie                    theMovie,
    Track                    theTrack,
    TimeValue                startTime,
    TimeValue                duration,
```

MovieExportNewGetDataAndPropertiesProcs

```
MovieExportGetPropertyUPP    *getPropertyProc,  
MovieExportGetDataUPP       *getDataProc,  
void                        **refCon );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

trackType

The format of the data to be generated by the returned `MovieExportGetDataProc` (III–2101).

scale

The time scale returned from this function; this should be passed on to `MovieExportAddDataSource` (II–919) with the procedures.

theMovie

The movie for this operation, supplied by the Movie Toolbox. Your component may use this identifier to obtain sample data from the movie or to obtain information about the movie.

theTrack

The track for this operation. This track identifier is supplied by the Movie Toolbox.

startTime

The starting point of the track or movie segment to be converted. This time value is expressed in the movie’s time coordinate system.

duration

The duration of the track or movie segment to be converted. This duration value is expressed in the movie’s time coordinate system.

getPropertyProc

A `MovieExportGetPropertyProc` (III–2102) callback that provides information about processing source samples.

getDataProc

A `MovieExportGetDataProc` (III–2101) callback that the export component uses to request sample data.

refCon

Passed to the procedures specified in the `getPropertyProc` and `getDataProc` parameters. Use this parameter to point to a data structure containing any information your callbacks need.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function exists in order to provide a standard way of getting data using this protocol out of a movie or track. The returned procedures must be disposed by calling `MovieExportDisposeGetDataAndPropertiesProcs` (II–920).

Special Considerations

This function is only implemented by movie data export components.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Movie Data” (V–2858)

MovieExportSetGetMoviePropertyProc

Specifies the procedure that the export component should call to retrieve movie level properties during `MovieExportFromProceduresToDataRef` (II–922).

```
ComponentResult MovieExportSetGetMoviePropertyProc (
    MovieExportComponent      ci,
    MovieExportGetPropertyUPP  getPropertyProc,
    void                      *refCon );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

getPropertyProc

The `MovieExportGetPropertyProc` (III–2102) callback that the export component will call to retrieve movie-level properties.

refCon

The reference value that will be passed to the callback specified by `getPropertyProc`. Use this parameter to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4. With QuickTime 4, applications can specify a `MovieExportGetPropertyProc` that will be called to retrieve movie level properties during the exporter’s `MovieExportFromProceduresToDataRef` (II–922) execution. This procedure is identical to a data source property procedure except that it is called for

MovieExportSetProgressProc

movie properties. For example, with QuickTime 4, the QuickTime movie export component calls the procedure to retrieve the time scale for the exported movie. If the property procedure is not specified or doesn't support this property, then the default movie time scale (600) is used.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Exporting Movie Data” (V–2858)

MovieExportSetProgressProc

Assigns a movie **progress function**.

```
ComponentResult MovieExportSetProgressProc (
    MovieExportComponent    ci,
    MovieProgressUPP        proc,
    long                    refcon );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

proc

A pointer to the application's `MovieProgressProc` (III–2105) callback. If this parameter is set to `NIL`, the application is removing its **progress function**. In this case, your component should stop calling the progress function.

refcon

A reference constant. Your component should pass this constant back to the application's **progress function** whenever you call that function. Use this parameter to point to a data structure containing any information the callback needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

These **progress functions** must support the same interface as Movie Toolbox progress functions. Note that this interface not only allows you to report progress to the application, but also allows the application to cancel the request.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Export Components” (V–2859)

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.setProgressProc()`

MovieExportSetSampleDescription

Requests the format of the exported data.

```
ComponentResult MovieExportSetSampleDescription (
    MovieExportComponent    ci,
    SampleDescriptionHandle desc,
    OSType                  mediaType );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

desc

A handle to a valid `SampleDescription` (IV–2414) structure.

mediaType

The type of media the `SampleDescription` (IV–2414) structure is for. For example, if the sample description was a sound description, this parameter would be set to `SoundMediaType`.

function result See “Error Codes” (IV–2663). Returns `badComponentSelector` if you should be passing a QT atom container (see discussion, below). Returns `noErr` if there is no error.

Discussion

A movie export component may use all, some, or none of the settings from the `SampleDescription` (IV–2414) structure.

If your application attempts to set the sample description using this function, and receives the `badComponentSelector` error, you may need to pass in the sample description using `MovieExportSetSettingsFromAtomContainer` (II–932). You can use `MovieExportGetSettingsAsAtomContainer` (II–925) to obtain a correctly formatted atom container to modify.

Special Considerations

This function is not implemented by all movie export components, but is supported by the sound movie export component, for example.

MovieExportSetSettingsFromAtomContainer

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Exporting Movie Data” (V–2858)

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.setSampleDescription()`

MovieExportSetSettingsFromAtomContainer

Sets the movie export component’s current configuration from passed settings data.

```
ComponentResult MovieExportSetSettingsFromAtomContainer (
    MovieExportComponent ci,
    QTAtomContainer settings );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

settings

A QT atom container that contains the settings.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The atom container may contain atoms other than those expected by the particular component type or may be missing certain atoms. This function uses only those settings it understands.

Here is sample code that overrides compression settings:

```
// MovieExportSetSettingsFromAtomContainer coding example
ComponentInstance sc;
QTAtomContainer compressorData;
SCSpatialSettings ss;
sc = OpenDefaultComponent(StandardCompressionType,
                          StandardCompressionSubType);
ss.codecType = kCinepakCodecType;
ss.codec = NIL;
ss.depth = 0;
ss.spatialQuality = codecHighQuality
```

```
err = SCSetInfo(sc, scSpatialSettingsType, &ss);
err = SCGetSettingsAsAtomContainer(sc, &compressorData);
MovieExportSetSettingsFromAtomContainer (qtvExport, compressorData);
```

Special Considerations

Some movie export components treat sample descriptions as part of their settings. If your application attempts to set the sample description using `MovieExportSetSampleDescription` (II–931), and receives the `badComponentSelector` error, you may need to pass in the `SampleDescription` (IV–2414) structure using this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Movie Data” (V–2858)

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.setExportSettingsFromAtomContainer`

See Also

You can use `MovieExportGetSettingsAsAtomContainer` (II–925) to obtain a correctly formatted atom container that you can then modify.

MovieExportToDataRef

Allows an application to request that data be exported to a data reference instead of to a file.

```
ComponentResult MovieExportToDataRef (
    MovieExportComponent    ci,
    Handle                  dataRef,
    OSType                  dataRefType,
    Movie                   theMovie,
    Track                   onlyThisTrack,
    TimeValue               startTime,
    TimeValue               duration );
```

`ci`

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`dataRef`

A handle to a data reference indicating where the data should be stored.

MovieExportToFile

`dataRefType`

The type of the data reference. For exporting to a file, the `dataRef` is a Macintosh file **alias** and the `dataRefType` is `rAliasType`.

`theMovie`

The movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain sample data from the movie or to obtain information about the movie.

`onlyThisTrack`

Identifies a track that is to be converted. This track identifier is supplied by the Movie Toolbox. If this parameter contains a track identifier, your component must convert only the specified track.

`startTime`

The starting point of the track or movie segment to be converted. This time value is expressed in the movie's time coordinate system.

`duration`

The duration of the track or movie segment to be converted. This duration value is expressed in the movie's time coordinate system.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Movie Data” (V–2858)

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.toDataRef()`

MovieExportToFile

Exports data to a file, using a movie data export component.

```
ComponentResult MovieExportToFile (
    MovieExportComponent    ci,
    const FSSpec             *theFile,
    Movie                   theMovie,
    Track                   onlyThisTrack,
    TimeValue               startTime,
    TimeValue               duration );
```

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

theFile

A pointer to the file that is to receive the converted movie data. This file's type value corresponds to your component's subtype value.

theMovie

The movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain sample data from the movie or to obtain information about the movie.

onlyThisTrack

Identifies a track that is to be converted. This track identifier is supplied by the Movie Toolbox. If this parameter contains a track identifier, your component must convert only the specified track.

startTime

The starting point of the track or movie segment to be converted. This time value is expressed in the movie's time coordinate system.

duration

The duration of the track or movie segment to be converted. This duration value is expressed in the movie's time coordinate system.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The requesting program or Movie Toolbox must create the destination file before calling this function. Furthermore, your component may not destroy any data in the destination file. If you cannot add data to the specified file, return an appropriate error. If your component can write data to a file, be sure to set the `canMovieExportFiles` flag in the `componentFlags` field of your component's `ComponentDescription` (IV–2212) structure. Here is an example of using this function with a flattener component:

```
// MovieExportToFile coding example
ComponentDescription desc;
Component flattener;
ComponentInstance qtvrExport = NIL;

desc.componentType = MovieExportType;
desc.componentSubType = MovieFileType;
desc.componentManufacturer = QTVRFlattenerType;
```

MovieExportToHandle

```
flattener = FindNextComponent(NIL, &desc);
if (flattener) qtvrExport = OpenComponent (flattener);

if (qtvrExport)
    MovieExportToFile (qtvrExport, &myFileSpec, myQTVRMovie, NIL, 0, 0);
```

Special Considerations

Your component must be prepared to perform this function at any time. You should not expect that any of your component's configuration functions will be called first.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: "Exporting Movie Data" (V-2858)

Related Java Methods

quicktime.std.qtcomponents.MovieExporter.toFile()

MovieExportToHandle

Exports data from a movie, using a movie data export component.

```
ComponentResult MovieExportToHandle (
    MovieExportComponent    ci,
    Handle                  dataH,
    Movie                   theMovie,
    Track                   onlyThisTrack,
    TimeValue               startTime,
    TimeValue               duration );
```

ci

A movie export component instance. Your software obtains this reference from OpenComponent (II-1130) or OpenDefaultComponent (II-1131).

dataH

A handle to be filled with the converted movie data. Your component must write data into this handle that corresponds to your component's subtype value. Your component should resize this handle as appropriate.

theMovie

The movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain sample data from the movie or to obtain information about the movie.

`onlyThisTrack`

Identifies a track that is to be converted. This track identifier is supplied by the Movie Toolbox. If this parameter contains a track identifier, your component must convert only the specified track.

`startTime`

The starting point of the track or movie segment to be converted. This time value is expressed in the movie's time coordinate system.

`duration`

The duration of the track or movie segment to be converted. This duration value is expressed in the movie's time coordinate system.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

Your component must be prepared to perform this function at any time. You should not expect that any of your component's configuration functions will be called first. If your component can write data to a handle, be sure to set the `canMovieExportHandles` flag in the `componentFlags` field of your component's `ComponentDescription` (IV-2212) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: "Exporting Movie Data" (V-2858)

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.toHandle()`

MovieExportValidate

Determines whether a movie export component can export all the data for a specified movie or track.

```
ComponentResult MovieExportValidate (
    MovieExportComponent    ci,
    Movie                   theMovie,
    Track                   onlyThisTrack,
    Boolean                  *valid );
```

MovieImportDataRef

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

theMovie

The movie to validate.

onlyThisTrack

A track within the movie to validate, or `NIL` if the entire movie is to be validated.

valid

A pointer to a `Boolean` value. If the data for the movie or track can be exported by the component, the value is `TRUE`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function allows an application to determine if a particular movie or track could be exported by the specified movie data export component. The movie or track is passed in the `theMovie` and `onlyThisTrack` parameters as they are passed to `MovieExportToFile` (II–934). Although a movie export component can export one or more media types, it may not be able to export all the kinds of data stored in those media. The `MovieExportValidate` function allows applications to get this additional information. Movie data export components that implement this function also set the `canMovieExportValidateMovie` flag in in the `componentFlags` field of their `ComponentDescription` (IV–2212) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Movie Data” (V–2858)

Related Java Methods

`quicktime.std.qtcomponents.MovieExporter.validate()`

MovieImportDataRef

Undocumented

```
ComponentResult MovieImportDataRef (
    MovieImportComponent    ci,
    Handle                  dataRef,
    OSType                  dataRefType,
    Movie                   theMovie,
```



```

Track          targetTrack,
Track          *usedTrack,
TimeValue      atTime,
TimeValue      *addedDuration,
long           inFlags,
long           *outFlags );

```

ci

A movie import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

dataRef

The data reference to the data to be imported.

dataRefType

The type of data reference in the `dataRef` parameter.

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

targetTrack

Undocumented

usedTrack

Undocumented

atTime

Undocumented

addedDuration

Undocumented

inFlags

Flags (see below) that control the behavior of this function.

outFlags

Flags (see below) that this function sets on return.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inFlags Constants

`movieImportCreateTrack`

Undocumented

`movieImportInParallel`

Undocumented

MovieImportDoUserDialog

movieImportMustUseTrack

Undocumented

movieImportWithIdle

Undocumented

outFlags Constants

movieImportResultUsedMultipleTracks

Undocumented

movieImportResultNeedIdles

Undocumented

movieImportResultComplete

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Related Java Methods

quicktime.std.movies.Track.fromMovieImporterDataRef(),

quicktime.std.qtcomponents.MovieImporter.fromDataRef()

MovieImportDoUserDialog

Requests that a component display its user dialog box.

```
ComponentResult MovieImportDoUserDialog (
    MovieImportComponent    ci,
    const FSSpec             *theFile,
    Handle                   theData,
    Boolean                  *canceled );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

theFile

A pointer to a valid file specification. If the import request pertains to a file, the application must specify the source file with this parameter and set the `theData` parameter to NIL. If the request is for a handle, this parameter is set to NIL.

theData

A handle to the data to be imported. If the import request pertains to a handle, the application must specify the source of the data with this parameter, and set the *theFile* parameter to `NIL`. If the request is for a file, this parameter is set to `NIL`.

canceled

A pointer to a `Boolean` value. Your component should set this value to `TRUE` if the user cancels the dialog box; otherwise, set it to `FALSE`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If your component supports a user dialog box, be sure to set the `hasMovieImportUserInterface` flag in the `componentFlags` field of your component’s `ComponentDescription` (IV–2212) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Import Components” (V–2856)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.doUserDialog()`

MovieImportEstimateCompletionTime

Undocumented

```
ComponentResult MovieImportEstimateCompletionTime (
    MovieImportComponent    ci,
    TimeRecord               *time );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

time

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

MovieImportFile

Programming Info

C interface file: `QuickTimeComponents.h`

MovieImportFile

Imports data from a file, using a movie data import component.

```
ComponentResult MovieImportFile (
    MovieImportComponent    ci,
    const FSSpec             *theFile,
    Movie                   theMovie,
    Track                   targetTrack,
    Track                   *usedTrack,
    TimeValue               atTime,
    TimeValue               *addedDuration,
    long                    inFlags,
    long                    *outFlags );
```

`ci`

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`theFile`

A pointer to the file that contains the data that is to be imported into the movie. This file's type value corresponds to your component's subtype value.

`theMovie`

The movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to add sample data to the target movie or to obtain information about the movie.

`targetTrack`

The track that is to receive the imported data. This track identifier is supplied by the Movie Toolbox and is valid only if the `movieImportMustUseTrack` flag in the `inFlags` parameter is set to 1.

`usedTrack`

A pointer to the track that received the imported data. Your component returns this track identifier to the Movie Toolbox. Your component needs to set this parameter only if you operate on a single track or if you create a new track. If you modify more than one track, leave the field referred to by this parameter unchanged.

`atTime`

The time corresponding to the location where your component is to place the imported data. This time value is expressed in the movie's time coordinate system.

`addedDuration`

A pointer to the duration of the data that your component added to the movie. Your component must specify this value in the movie's time coordinate system.

`inFlags`

Flags (see below) that specify control information governing the import operation.

`outFlags`

Flags (see below) that identify a field that is to receive status information about the import operation. Your component sets the appropriate flags in this field when the operation is complete.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

inFlags Constants

`movieImportCreateTrack`

Indicates that your component should create a new track to receive the imported data. You must create a track whose type value corresponds to the media type you have specified in your component's manufacturer code. You should return the track identifier of this new track in the field referred to by the `usedTrack` parameter, unless you create more than one track. If you create more than one track, be sure to set the `movieImportResultUsedMultipleTracks` flag (in the field referred to by the `outFlags` parameter) to 1. If the `movieImportCreateTrack` flag is set to 1, then the `movieImportMustUseTrack` flag is set to 0.

`movieImportMustUseTrack`

Indicates that your component must use an existing track. That track is identified by the `targetTrack` parameter. If you create more than one track, be sure to set the `movieImportResultUsedMultipleTracks` flag (in the field referred to by the `outFlags` parameter) to 1. If the `movieImportMustUseTrack` flag is set to 1, then the `movieImportCreateTrack` flag is set to 0. If both the `movieImportCreateTrack` and `movieImportMustUseTrack` flags are set to 0, then you are free to use any existing tracks in the movie, or to create a new track (or tracks) as needed.

`movieImportInParallel`

Indicates whether you are to perform an insert operation or a paste operation. If this flag is set to 0, then you should insert the imported data into the target



MovieImportGetAuxiliaryDataType

track. If this flag is set to 1, then you should add the imported data to the track, overwriting the preexisting open space currently in the track. Note that an application may use `MovieImportSetDuration` (II–957) to control the amount of data you paste into a movie. If the `movieImportMustUseTrack` flag is set to 1, then you should use the track specified by the `targetTrack` parameter. If this is not possible, return an appropriate Movie Toolbox result code.

outFlags Constants

`movieImportResultUsedMultipleTracks`

Indicates that your component modified more than one track in the movie. Set this flag to 1 if your component places imported data into more than one track. In this case, you do not need to update the field referred to by the `usedTrack` parameter.

`movieImportInParallel`

Indicates whether you performed an insert operation or a paste operation. Set this flag to 0 if you inserted the imported data into the target track. Set this flag to 1 if you added the imported data to the track, overwriting preexisting open space currently in the track.

Discussion

Your component must be prepared to perform this function at any time. You should not expect that any of your component’s configuration functions will be called first. If your component can accept data from a file, be sure to set the `canMovieImportFiles` flag in the `componentFlags` field of your component’s `ComponentDescription` (IV–2212) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Importing Movie Data” (V–2855)

Related Java Methods

```
quicktime.std.movies.Track.fromMovieImporterFile(),  
quicktime.std.qtcomponents.MovieImporter.fromFile()
```

MovieImportGetAuxiliaryDataType

Returns the type of the auxiliary data that a component can accept.

```
ComponentResult MovieImportGetAuxiliaryDataType (  
    MovieImportComponent    ci,  
    OSType                  *auxType );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

auxType

A pointer to the type of auxiliary data it can import.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns the type of the auxiliary data that the *ci* component can accept. For example, calling the text import component with this function indicates that the text import component will use 'styl' information in addition to 'TEXT' data. Note that if component includes a private component **resource** holding this MIME data, it can use `GetComponentResource` (I–394) to retrieve it. If the resource is a public component resource, it either use `GetComponentPublicResource` (I–392) with the public type and ID or `GetComponentResource` with the private type and ID.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Importing Movie Data” (V–2855)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.getAuxiliaryDataType()`

MovieImportGetDontBlock

Undocumented

```
ComponentResult MovieImportGetDontBlock (
    MovieImportComponent    ci,
    Boolean                  *willBlock );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

willBlock

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

MovieImportGetFileType

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

MovieImportGetFileType

Allows your movie data import component to tell the Movie Toolbox the appropriate file type for the most-recently imported movie file.

```
ComponentResult MovieImportGetFileType (
    MovieImportComponent ci,
    OSType *fileType );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

fileType

A pointer to an `OSType` field. Your component should place the file type value that best identifies the movie data just imported. For example, Apple’s Audio CD movie data import component sets this field to ‘AIFF’ whenever it creates an AIFF file instead of a movie file.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You should implement this function only if your movie data import component creates files other than QuickTime movies. By default, the Movie Toolbox makes new files into movies, unless you override that default by providing this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Importing Movie Data” (V–2855)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.getFileType()`

MovieImportGetLoadState

Undocumented

```
ComponentResult MovieImportGetLoadState (
    MovieImportComponent    ci,
    long                    *importerLoadState );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

importerLoadState

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `QuickTimeComponents.h`

MovieImportGetMaxLoadedTime

Undocumented

```
ComponentResult MovieImportGetMaxLoadedTime (
    MovieImportComponent    ci,
    TimeValue               *time );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

time

A pointer to a value containing the maximum loaded time.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `QuickTimeComponents.h`

MovieImportGetMIMETypeList

MovieImportGetMIMETypeList

Returns a list of MIME types supported by the movie import component.

```
ComponentResult MovieImportGetMIMETypeList (  
    MovieImportComponent    ci,  
    QTAtomContainer         *mimeInfo );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

mimeInfo

A pointer to a MIME type list, a QT atom container that contains a list of MIME types supported by the movie import component. The caller should dispose of the atom container when finished with it.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your movie import component can support MIME types that correspond to formats it supports. To make a list of these MIME types available to applications or other software, it must implement this function. To indicate that your movie import component supports this function, set the `hasMovieImportMIMEList` flag in the `componentFlags` field of the `ComponentDescription` (IV–2212) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Importing Movie Data” (V–2855)

Related Java Methods

```
quicktime.std.qtcomponents.MovieImporter.getMIMETypeList(),  
quicktime.std.movies.AtomContainer.fromMovieImporterMIME()
```

See Also

If your component includes a private component **resource** holding the MIME data, it can use `GetComponentResource` (I–394) to retrieve it. If the resource is a public component resource, it can use either `GetComponentPublicResource` (I–392) with the public type and ID or `GetComponentResource` with the private type and ID.

MovieImportGetSampleDescription

Gets the current sample description for a movie import component.

```
ComponentResult MovieImportGetSampleDescription (
    MovieImportComponent    ci,
    SampleDescriptionHandle  *desc,
    OSType                  *mediaType );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

desc

A pointer to a handle to a `SampleDescription` (IV–2414) structure.

mediaType

A pointer to the type of the data; see “Data References” (IV–2661).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Related Java Methods

```
quicktime.std.movies.media.SampleDescription.fromMovieImporter(),
quicktime.std.qtcomponents.MovieImporter.getSampleDescription(),
quicktime.std.qtcomponents.MovieImporter.getMediaType()
```

See Also

For the corresponding set function, see `MovieImportSetSampleDescription` (II–961).

MovieImportGetSettingsAsAtomContainer

Retrieves the current settings from the movie import component.

```
ComponentResult MovieImportGetSettingsAsAtomContainer (
    MovieImportComponent    ci,
    QTAtomContainer         *settings );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

MovieImportHandle

settings

The address where the reference to the newly created atom container should be stored by the call.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The caller is responsible for disposing of the returned QT atom container.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Importing Movie Data” (V–2855)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.getImportSettingsFromAtomContainer`

`quicktime.std.movies.AtomContainer.fromMovieImporterSettings()`

MovieImportHandle

Imports data from a handle, using a movie data import component.

```
ComponentResult MovieImportHandle (
    MovieImportComponent    ci,
    Handle                  dataH,
    Movie                   theMovie,
    Track                   targetTrack,
    Track                   *usedTrack,
    TimeValue               atTime,
    TimeValue               *addedDuration,
    long                    inFlags,
    long                    *outFlags );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

dataH

A handle to the data that is to be imported into the movie identified by the `theMovie` parameter. The data contained in this handle has a data type value that corresponds to your component’s subtype value. Your component is not responsible for disposing of this handle.

`theMovie`

The movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to add sample data to the target movie, or to obtain information about the movie.

`targetTrack`

The track that is to receive the imported data. This track identifier is supplied by the Movie Toolbox and is valid only if the `movieImportMustUseTrack` flag in the `inFlags` parameter is set to 1.

`usedTrack`

A pointer to the track that received the imported data. Your component returns this track identifier to the Movie Toolbox. Your component needs to set this parameter only if you operate on a single track or if you create a new track. If you modify more than one track, leave the field referred to by this parameter unchanged.

`atTime`

The time corresponding to the location where your component is to place the imported data. This time value is expressed in the movie's time coordinate system.

`addedDuration`

A pointer to the duration of the data that your component added to the movie. Your component must specify this value in the movie's time coordinate system.

`inFlags`

Flags (see below) that specify control information governing the import operation.

`outFlags`

Flags (see below) that receive status information about the import operation. Your component sets the appropriate flags in this field when the operation is complete.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

inFlags Constants

`movieImportCreateTrack`

Indicates that your component should create a new track to receive the imported data. You must create a track whose type value corresponds to the media type that you have specified in your component's manufacturer code. You should return the track identifier of this new track in the field referred to by the `usedTrack` parameter, unless you create more than one track. If you create more than one track, be sure to set the `movieImportResultUsedMultipleTracks` flag in the field referred to by the `outFlags` parameter to 1. If the



MovieImportHandle

`movieImportCreateTrack` flag is set to 1, then the `movieImportMustUseTrack` flag is set to 0.

`movieImportMustUseTrack`

Indicates that your component must use an existing track. That track is identified by the `targetTrack` parameter. If you create more than one track, be sure to set the `movieImportResultUsedMultipleTracks` flag in the field referred to by the `outFlags` parameter to 1. If the `movieImportMustUseTrack` flag is set to 1, then the `movieImportCreateTrack` flag is set to 0. If both the `movieImportCreateTrack` and `movieImportMustUseTrack` flags are set to 0, then you are free to use any existing tracks in the movie or to create a new track (or tracks) as needed.

`movieImportInParallel`

Indicates whether you are to perform an insert operation or a paste operation. If this flag is set to 0, then you should insert the imported data into the target track. If this flag is set to 1, then you should add the imported data to the track, overwriting preexisting open space currently in the track. Note that an application may use `MovieImportSetDuration` (II-957) to control the amount of data you paste into a movie. If the `movieImportMustUseTrack` flag is set to 1, then you should use the track specified by the `targetTrack` parameter. If this is not possible, return an appropriate Movie Toolbox result code.

outFlags Constants

`movieImportResultUsedMultipleTracks`

Indicates that your component modified more than one track in the movie. Set this flag to 1 if your component places imported data into more than one track. In this case, you do not need to update the field referred to by the `usedTrack` parameter.

`movieImportInParallel`

Indicates whether you performed an insert operation or a paste operation. Set this flag to 0 if you inserted the imported data into the target track. Set this flag to 1 if you added the imported data to the track, overwriting preexisting open space currently in the track.

Discussion

Your component must be prepared to perform this function at any time. You should not expect that any of your component's configuration functions will be called first. If your component can accept data from a handle, be sure to set the `canMovieImportHandles` flag in your component's `componentFlags` field.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`
 Programming summary: “Importing Movie Data” (V–2855)

Related Java Methods

`quicktime.std.movies.Track.fromMovieImporterHandle()`,
`quicktime.std.qtcomponents.MovieImporter.fromHandle()`

MovieImportIdle

Undocumented

```
ComponentResult MovieImportIdle (
    MovieImportComponent    ci,
    long                    inFlags,
    long                    *outFlags );
```

`ci`

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`inFlags`

Undocumented

`outFlags`

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inFlags Constants

Undocumented

Undocumented

outFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`



MovieImportSetAuxiliaryData

MovieImportSetAuxiliaryData

Provides additional data to a component.

```
ComponentResult MovieImportSetAuxiliaryData (
    MovieImportComponent    ci,
    Handle                  data,
    OSType                  handleType );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

data

A handle to the additional data. Your component should not dispose of this handle. Be sure to copy any data you need to keep.

handleType

The type of data in the specified handle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your component should expect the application to call this function before the import process begins.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Import Components” (V–2856)

Related Java Methods

```
quicktime.std.qtcomponents.MovieImporter.setAuxiliaryData()
```

MovieImportSetChunkSize

The amount of data a component works with at a time.

```
ComponentResult MovieImportSetChunkSize (
    MovieImportComponent    ci,
    long                    chunkSize );
```


`ci`

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`chunkSize`

The number of seconds of data your movie data import component places into each chunk of movie data. This parameter may not be set to a value less than 1.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Generally, your component should determine a reasonable default chunk size, based on the type of data you are importing. However, you may choose to allow applications to override your default value. This can be especially useful for sound data, where the chunk size affects the quality of sound playback.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Import Components” (V–2856)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.setChunkSize()`

MovieImportSetDimensions

Specifies a new track’s spatial dimensions.

```
ComponentResult MovieImportSetDimensions (
    MovieImportComponent ci,
    Fixed width,
    Fixed height );
```

`ci`

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`width`

The width, in pixels, of the track rectangle. This parameter, along with the `height` parameter, specifies a rectangle that surrounds the image that is to be displayed when the current media is played. This value corresponds to the x coordinate of the lower-right corner of the rectangle, and it is expressed as a fixed-point number.



MovieImportSetDontBlock

height

The height, in pixels, of the track rectangle. This value corresponds to the y coordinate of the lower-right corner of the rectangle, and it is expressed as a fixed-point number.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Import Components” (V–2856)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.setDimensions()`

See Also

If you want to change the track’s matrix, call `SetTrackMatrix` (III–1662) after performing the import operation.

MovieImportSetDontBlock

Undocumented

```
ComponentResult MovieImportSetDontBlock (
    MovieImportComponent    ci,
    Boolean                  dontBlock );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

dontBlock

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

MovieImportSetDuration

Controls the duration of the data that a component pastes into the target movie.

```
ComponentResult MovieImportSetDuration (
    MovieImportComponent    ci,
    TimeValue                duration );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

duration

The duration in the movie’s time scale. If this parameter is set to 0, then you may paste any amount of movie data that is appropriate for the data to be imported.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If your component supports paste operations (that is, your component allows the application to set the `movieImportInParallel` flag to 1 with the `MovieImportHandle` (II–950) or `MovieImportFile` (II–942) function), then you must support this function. If an application calls this function and sets a duration limit, you must abide by that limit. This function is not valid for insert operations (where the `movieImportInParallel` flag is set to 0).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Import Components” (V–2856)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.setDuration()`

MovieImportSetFromScrap

Indicates that the source data resides on the scrap.

```
ComponentResult MovieImportSetFromScrap (
    MovieImportComponent    ci,
    Boolean                  fromScrap );
```

MovieImportSetMediaFile

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

fromScrap

Set to `TRUE` if the data originated on the scrap; otherwise, set to `FALSE`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Import Components” (V–2856)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.setFromScrap()`

MovieImportSetMediaFile

Specifies a media file that is to receive the imported movie data.

```
ComponentResult MovieImportSetMediaFile (  
    MovieImportComponent    ci,  
    AliasHandle              alias );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

alias

The media file that is to receive the imported movie data. Your component must make a copy of this parameter. You should not dispose of it.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Import Components” (V–2856)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.setMediaFile()`

MovieImportSetOffsetAndLimit

Specifies location and size of data that should be imported.

```
ComponentResult MovieImportSetOffsetAndLimit (
    MovieImportComponent    ci,
    unsigned long            offset,
    unsigned long            limit );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

offset

A byte offset into the file that indicates where the import operation begins.

limit

A byte offset into the file that indicates the last data in the file that can be imported.

function result See “Error Codes” (IV–2663). Returns `badComponentSelector` if the movie import component does not support this function. Returns `noErr` if there is no error.

Discussion

Typically, this function is used when the data is from a part of a file rather than the entire file. It is especially useful when one file format is embedded in another; it allows your application to skip header data for the enclosing file and begin importing data at the start of the desired format.

Special Considerations

Not all movie import components support this function. Those that do include the movie import components for the `kQTFileTypeAIFF`, `kQTFileTypeWave`, and `kQTFileTypeMuLaw` file types. Those that do not return the `badComponentSelector` result code in response to a this call.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Importing Movie Data” (V–2855)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.setOffsetAndLimit()`

MovieImportSetOffsetAndLimit64

MovieImportSetOffsetAndLimit64

Specifies location and size of data that should be imported from a file.

```
ComponentResult MovieImportSetOffsetAndLimit64 (  
    MovieImportComponent    ci,  
    const wide               *offset,  
    const wide               *limit );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

offset

A byte offset into the file that indicates where the import operation begins.

limit

A byte offset into the file that indicates the last data in the file that can be imported.

function result See “Error Codes” (IV–2663). Returns `badComponentSelector` if the movie import component does not support this function. Returns `noErr` if there is no error.

Discussion

This function serves the same purpose as `MovieImportSetOffsetAndLimit` (II–959). The only difference is that the offset and limit can hold 64-bit offsets. This function is especially useful when one file format is embedded in another; it allows your application to skip header data for the enclosing file and begin importing data at the start of the desired format.

Special Considerations

Not all movie import components support this function. Those that do not return the `badComponentSelector` result code. If this function is not implemented and the offset and limit can be expressed using 32-bit offsets, `MovieImportSetOffsetAndLimit` (II–959) should be tried.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Importing Movie Data” (V–2855)

MovieImportSetProgressProc

Assigns a movie **progress function**.

```
ComponentResult MovieImportSetProgressProc (
    MovieImportComponent    ci,
    MovieProgressUPP        proc,
    long                    refcon );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

proc

A pointer to the application’s `MovieProgressProc` (III–2105) callback. If this parameter is set to `NIL`, the application is removing its **progress function**. In this case, your component should stop calling the progress function.

refcon

Specifies a reference constant. Your component should pass this constant back to the application’s **progress function** whenever you call that function. The application may use this parameter to point to a data structure containing any information the callback needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The `MovieProgressProc` (III–2105) callback interface not only allows you to report progress to the application, but also allows the application to cancel the request.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Import Components” (V–2856)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.setProgressProc()`

MovieImportSetSampleDescription

Provides a `SampleDescription` (IV–2414) structure to a movie data import component.

```
ComponentResult MovieImportSetSampleDescription (
```

MovieImportSetSampleDuration

```
MovieImportComponent    ci,  
SampleDescriptionHandle desc,  
OSType                  mediaType );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

desc

A handle to a `SampleDescription` (IV–2414) structure. Your component must not dispose of this handle. If you want to save any data from the structure, be sure to copy it at this time.

mediaType

The type of sample description referred to by the *desc* parameter. If the *desc* parameter refers to an `ImageDescription` (IV–2285) structure, this parameter is set to `VideoMediaType` ('vide'); for `SoundDescription` (IV–2449) structures, this parameter is set to `SoundMediaType` ('soun').

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Import Components” (V–2856)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.setSampleDescription()`

See Also

For the corresponding get function, see `MovieImportGetSampleDescription` (II–949).

MovieImportSetSampleDuration

Sets the sample duration for new samples to be created with a component.

```
ComponentResult MovieImportSetSampleDuration (  
    MovieImportComponent    ci,  
    TimeValue                duration,  
    TimeScale                scale );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

duration

The sample duration in units specified by the `scale` parameter.

scale

The time scale for the duration value. This may be any arbitrary time scale; that is, it may not correspond to the movie's time scale. You should convert this time scale to the movie's time scale before using the duration value, using `ConvertTimeScale` (I–138).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Movie Data Import Components” (V–2856)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.setSampleDuration()`

MovieImportSetSettingsFromAtomContainer

Sets the movie import component's current configuration from the passed settings data.

```
ComponentResult MovieImportSetSettingsFromAtomContainer (  
    MovieImportComponent    ci,  
    QTAtomContainer         settings );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

settings

A QT atom container containing settings.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The settings QT atom container may contain atoms other than those expected by the particular component type or may be missing certain atoms. The function uses only those settings it understands.

Version Notes

Introduced in QuickTime 3 or earlier.

MovieImportValidate

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Importing Movie Data” (V–2855)

Related Java Methods

quicktime.std.qtcomponents.MovieImporter.setImportSettingsFromAtomContainer

MovieImportValidate

Allows your movie data import component to validate the data to be passed to your component.

```
ComponentResult MovieImportValidate (
    MovieImportComponent    ci,
    const FSSpec             *theFile,
    Handle                   theData,
    Boolean                  *valid );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

theFile

An `FSSpec` (IV–2262) structure that defines the file to validate if the importer imports from files.

theData

The data to validate if the importer imports from handles.

valid

A pointer to a `Boolean` value. If the data or file can be imported, the value returned is `TRUE`. Otherwise, it returns `FALSE`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Movie import components can implement this function to allow applications to determine if a given file or handle to data is acceptable for a particular import component. As this function may be called on many files, the validation process should be as fast as possible.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Importing Movie Data” (V–2855)

Related Java Methods

`quicktime.std.qtcomponents.MovieImporter.validate()`

See Also

See `MovieImportValidateDataRef` (II–965).

MovieImportValidateDataRef

Validates the data file indicated by the data reference.

```
ComponentResult MovieImportValidateDataRef (
    MovieImportComponent    ci,
    Handle                   dataRef,
    OSType                   dataRefType,
    UInt8                    *valid );
```

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

dataRef

The data reference to the file to be validated.

dataRefType

The type of data reference for the *dataRef* parameter.

valid

A pointer to a `UInt8` value. If the data or file cannot be imported, the value returned should be 0. Otherwise, it should be set to 128.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Movie import components can implement this function to allow applications to determine if a given file referenced by a data reference is acceptable for a particular import component. The data reference can refer to any data for which there is a suitable data handler component installed and available to QuickTime. As this function may be called on many files, the validation process should be as fast as possible. Furthermore, the importer should probably limit the amount of reading it performs, especially when the data handler refers to data on the Internet.

MovieMediaGetChildDoMCActionCallback

Special Considerations

Unlike `MovieImportValidate` (II–964), the `valid` parameter for this function is a value that can be interpreted as the degree to which the importer can interpret the file’s contents. In all cases, returning 0 indicates the file cannot be interpreted by the importer. However, other non-zero values can be used to determine the relative weighting between multiple importers that can import a particular kind of file. For now, it is best to return either 0 or 128 only.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Importing Movie Data” (V–2855)

See Also

See `MovieImportValidate` (II–964).

MovieMediaGetChildDoMCActionCallback

Undocumented

```
ComponentResult MovieMediaGetChildDoMCActionCallback (
    MediaHandler      mh,
    DoMCActionUPP     *doMCActionCallbackProc,
    long              *refcon );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

doMCActionCallbackProc

A pointer to a **Universal Procedure Pointer** that accesses a `DoMCActionProc` (III–2082) callback.

refcon

A pointer to a reference constant to be passed to your callback. Use this constant to point to a data structure containing any information your callback needs.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

MovieMediaGetChildMovieDataReference

Undocumented

```
ComponentResult MovieMediaGetChildMovieDataReference (
    MediaHandler      mh,
    QTAtomID          dataRefID,
    short             dataRefIndex,
    OSType            *dataRefType,
    Handle            *dataRef,
    QTAtomID          *dataRefIDOut,
    short             *dataRefIndexOut );
```

`mh`

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

`dataRefID`

Undocumented

`dataRefIndex`

Undocumented

`dataRefType`

Undocumented

`dataRef`

Undocumented

`dataRefIDOut`

Undocumented

`dataRefIndexOut`

Undocumented

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`



MovieMediaGetCurrentMovieProperty

See Also

For the corresponding set function, see `MovieMediaSetChildMovieDataReference` (II–971).

MovieMediaGetCurrentMovieProperty

Retrieves current properties from a media handler’s movie.

```
ComponentResult MovieMediaGetCurrentMovieProperty (  
    MediaHandler    mh,  
    OSType          whichProperty,  
    void            *value );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

whichProperty

A constant (see below) that designates the property to be retrieved.

value

A pointer to the returned property value.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

whichProperty Constants

`kMoviePropertyDuration`

The movie’s duration. The value of the constant is 'dura'.

`kMoviePropertyTimeScale`

The movie’s time scale. The value of the constant is 'tims'.

`kMoviePropertyTime`

The movie’s current time. The value of the constant is 'timv'.

`kMoviePropertyNaturalBounds`

The movie’s boundary rectangle, returned as a `Rect` (IV–2399) structure. The value of the constant is 'natb'.

`kMoviePropertyMatrix`

The movie’s matrix, returned as a `MatrixRecord` (IV–2304) structure. The value of the constant is 'mtrx'.

`kMoviePropertyTrackList`

A pointer to the movie’s track list, returned as a long integer. The value of the constant is 'tlst'.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

MovieMediaGetCurrentTrackProperty

Retrieves the media type property from a media handler's track.

```
ComponentResult MovieMediaGetCurrentTrackProperty (
    MediaHandler    mh,
    long            trackID,
    OSType          whichProperty,
    void            *value );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I-432).

trackID

The ID value of the track for this operation.

whichProperty

A constant (see below) that designates the property to be retrieved. Only the track's media type property constant is currently defined.

value

A pointer to the returned property value.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

whichProperty Constant

`kTrackPropertyMediaType`

The track's media type, returned as an `OSType` value. The value of the constant is `'mtyp'`.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

MovieMediaGetDoMCActionCallback

MovieMediaGetDoMCActionCallback

Gets a DoMCActionProc (III–2082) callback for a media.

```
ComponentResult MovieMediaGetDoMCActionCallback (
    MediaHandler      mh,
    DoMCActionUPP     *doMCActionCallbackProc,
    long               *refcon );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

doMCActionCallbackProc

A pointer to a **Universal Procedure Pointer** that accesses a DoMCActionProc (III–2082) callback.

refcon

A pointer to a reference constant to be passed to your callback. Use this constant to point to a data structure containing any information your callback needs.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

MovieMediaLoadChildMovieFromDataReference

Undocumented

```
ComponentResult MovieMediaLoadChildMovieFromDataReference (
    MediaHandler      mh,
    QTAtomID          dataRefID );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

dataRefID

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function

result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

MovieMediaSetChildMovieDataReference

Undocumented

```
ComponentResult MovieMediaSetChildMovieDataReference (
    MediaHandler      mh,
    QTAtomID          dataRefID,
    OSType            dataRefType,
    Handle             dataRef );
```

mh

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

dataRefID

Undocumented

dataRefType

Undocumented

dataRef

Undocumented

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

See Also

For the corresponding get function, see `MovieMediaGetChildMovieDataReference` (II–967).



MovieSearchText

MovieSearchText

Searches for text in a movie.

```
OSErr MovieSearchText (
    Movie      theMovie,
    Ptr        text,
    long       size,
    long       searchFlags,
    Track      *searchTrack,
    TimeValue  *searchTime,
    long       *searchOffset );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

text

The text to be searched for.

size

The size of the text.

searchFlags

Flags (see below) that narrow the search process.

searchTrack

On return, a pointer to the found track.

searchTime

On return, a pointer to the found time.

searchOffset

On return, a pointer to the found offset to the text.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

searchFlags Constants

searchTextDontGoToFoundTime

By default, `MovieSearchText` automatically moves to the sample containing the found text and highlights it; this flag and `searchTextDontHiliteFoundText` override those default behaviors.

searchTextDontHiliteFoundText

Don’t highlight the found text.

`searchTextOneTrackOnly`

Search the track designated by `searchTrack` only.

`searchTextEnabledTracksOnly`

Search only enabled tracks.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.Movie.searchText()`

MoviesTask



Services active movies.

```
void MoviesTask (
    Movie    theMovie,
    long     maxMilliSecToUse );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084). If you set this parameter to `NIL`, the Movie Toolbox services all of your active movies.

`maxMilliSecToUse`

Determines the maximum number of milliseconds that `MoviesTask` can work before returning. If this parameter is 0, `MoviesTask` services every active movie exactly once and then returns. If the parameter is nonzero, `MoviesTask` services as many movies as it can in the allotted time before returning. Once the `MoviesTask` function starts servicing a movie, it cannot stop until it has completely met the requirements of the movie. Consequently, the `MoviesTask` function may execute for a longer time than that specified in `maxMilliSecToUse`. However, the function does not start servicing a new movie if the time specified by `maxMilliSecToUse` has elapsed. The preferred way to use `MoviesTask` is to set the `maxMilliSecToUse` parameter to 0; however, if you just want to play one movie, you can call `MoviesTask` on that one. If your rate is 0, `MoviesTask` draws that frame and no other.

function result You can access error returns from this function through

MusicDerivedCloseResFile

GetMoviesError (I–492) and GetMoviesStickyError (I–494). See “Error Codes” (IV–2663).

Discussion

You should call `MoviesTask` as often as possible from your application’s main event loop. Note that you should call this function after you have performed your own event processing. `MoviesTask` services only active movies, and only enabled tracks within those active movies.

Special Considerations

Note that the `MoviesTask` function services only your movies. Your application must give other applications the opportunity to call `MoviesTask` for their movies.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movies and Your Event Loop” (V–2720)

Related Java Methods

`quicktime.std.movies.Movie.taskAll()`, `quicktime.std.movies.Movie.task()`

See Also

Use `SetMovieActive` (III–1609) and `SetTrackEnabled` (III–1658) to enable and disable movies and tracks.

MusicDerivedCloseResFile

Closes a music movie **resource** file.

```
ComponentResult MusicDerivedCloseResFile (  
    MusicComponent    mc,  
    short              resRefNum );
```

`mc`

A music component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`resRefNum`

The **resource** file to be closed. Your application obtains this value from the `OpenMovieFile` (II–1133) function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

MusicDerivedMIDISend

Sends a MIDI packet to a music component.

```
ComponentResult MusicDerivedMIDISend (
    MusicComponent    mc,
    MusicMIDIPacket    *packet );
```

mc

A music component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

packet

A pointer to the music MIDI packet to be sent.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

MusicDerivedOpenResFile

Opens the music **resource** file for a music component.

```
ComponentResult MusicDerivedOpenResFile (
    MusicComponent    mc );
```

mc

A music component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

MusicDerivedSetInstrument

Programming Info

C interface file: QuickTimeMusic.h

MusicDerivedSetInstrument

The complete instrument defined by the Part structure to the synthesizer.

```
ComponentResult MusicDerivedSetInstrument (
    MusicComponent    mc,
    long              partNumber,
    GCPart             *p );
```

mc

The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

partNumber

The number of the part for this operation.

p

A pointer to the part for this operation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Calling Generic Music Component Clients” (V–2925)

MusicDerivedSetKnob

Called when any of the synthesizer’s knobs are altered.

```
ComponentResult MusicDerivedSetKnob (
    MusicComponent    mc,
    long              knobType,
    long              knobNumber,
    long              knobValue,
    long              partNumber,
    GCPart             *p,
    GenericKnobDescription *gkd );
```

mc

The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

knobType

The type of knob that has been altered (see below).

knobNumber

The number of the knob that has been altered.

knobValue

The new value of the altered knob.

partNumber

The number of the part whose knob has been altered.

p

A pointer to the part whose knob has been altered.

gkd

A `GenericKnobDescription` (IV–2267) structure for the knob.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

knobType Constants

`kGenericMusicKnob`

Value is 1.

`kGenericMusicInstrumentKnob`

Value is 2.

`kGenericMusicDrumKnob`

Value is 3.

Discussion

This function is called when any knob on the synthesizer is altered. It should look at the `GCPart` (IV–2263) and the `GenericKnobDescription` (IV–2267) structures and address the synthesizer hardware appropriately to set the new knob value. For a MIDI device, this means to construct a system-exclusive MIDI packet and send it to the MIDI routine received by the `MusicDerivedSetMIDI` (II–978) call.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Calling Generic Music Component Clients” (V–2925)

MusicDerivedSetMIDI

MusicDerivedSetMIDI

Sets the MIDI channel and other MIDI settings for MIDI output only.

```
ComponentResult MusicDerivedSetMIDI (  
    MusicComponent      mc,  
    MusicMIDISendUPP    midiProc,  
    long                refcon,  
    long                midiChannel );
```

mc

The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

midiProc

A pointer to the `MusicMIDISendProc` (III–2110) callback in your music component for performing MIDI output.

refcon

A reference constant sent to the callback specified by the `midiProc` parameter. Use this parameter to point to a data structure containing any information your callback needs.

midiChannel

The MIDI channel to use for the operation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

A derived component for a MIDI synthesizer receives this call soon after it is opened. It should store the `midiProc`, `refCon`, and `midiChannel` parameters in its global variables. When the derived component needs to communicate with the synthesizer, it calls your `MusicMIDISendProc` (III–2110) function with this reference constant. The `midiChannel` variable specifies the system channel of the device.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Calling Generic Music Component Clients” (V–2925)

MusicDerivedSetPart

Sets the polyphony for the part specified in the GCPart (IV–2263) structure.

```
ComponentResult MusicDerivedSetPart (
    MusicComponent    mc,
    long              partNumber,
    GCPart             *p );
```

mc

The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

partNumber

The number of the part for this operation.

p

A pointer to the part for this operation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Calling Generic Music Component Clients” (V–2925)

MusicDerivedSetPartInstrumentNumber

Sets the instrument specified in the GCPart (IV–2263) structure.

```
ComponentResult MusicDerivedSetPartInstrumentNumber (
    MusicComponent    mc,
    long              partNumber,
    GCPart             *p );
```

mc

A music component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

partNumber

The number of the part for this operation.

MusicDerivedStorePartInstrument

p

A pointer to the part for this operation.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

MusicDerivedStorePartInstrument

Undocumented

```
ComponentResult MusicDerivedStorePartInstrument (
    MusicComponent    mc,
    long               partNumber,
    GCPart             *p,
    long               instrumentNumber );
```

mc

An instance of the music component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

partNumber

The number of the part for this operation.

p

A pointer to the part for this operation.

instrumentNumber

Number of the instrument for this part. You can use `MusicFindTone` (II–981) to get an instrument number.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

MusicFindTone

Returns the number of the best-matching instrument provided by a specified music component.

```
ComponentResult MusicFindTone (
    MusicComponent    mc,
    ToneDescription    *td,
    long               *libraryIndexOut,
    unsigned long      *fit );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II-1028).

td

Pointer to a `ToneDescription` (IV-2487) structure.

libraryIndexOut

On return, contains the number of the best-matching instrument. Only General MIDI numbers are guaranteed to be the same for later instantiations of the component.

fit

On return, a constant (see below) that indicates how well an instrument matches the tone description.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

fit Constants

`kInstrumentMatchSynthesizerType`

The requested synthesizer type was found.

`kInstrumentMatchSynthesizerName`

The particular instance of the synthesizer requested was found.

`kInstrumentMatchName`

The instrument name in the tone description matched an appropriate instrument on the synthesizer.

`kInstrumentMatchNumber`

The instrument number in the tone description matched an appropriate instrument on the synthesizer.

`kInstrumentMatchGMNumber`

The General MIDI equivalent was used to find an appropriate instrument on the synthesizer.

MusicGenericConfigure

Discussion

The music component searches for an instrument as follows:

If the `synthesizerType` field of the `td` parameter matches the type of the specified music component, it first tries to find an instrument that matches the value of the `instrumentNumber` field of the `td` parameter. If this value is in the range 129–16512, which specifies a GS instrument, and the GS instrument is not available, it tries to find the General MIDI instrument that corresponds to it, which has the number $((\text{GSinstrumentnumber} - 1) \& 0x7F) + 1$. If the value is greater than 16512, which specifies a transient ROM instrument or internal instrument index value, it tries to find an instrument that matches the `synthesizerName` field of the `td` parameter. If that fails, it tries to find an instrument that matches the value of the `gmNumber` field of the `td` parameter.

If the `synthesizerType` field of the `td` parameter does not match the type of the specified music component, it tries to find an instrument that matches the value of the `gmNumber` field of the `td` parameter.

If none of these rules apply, or the fields are blank (0 for the type or numeric fields, or zero-length for the strings), then the call returns instrument 1 and a fit parameter of zero.

The `synthesizerName` field may be ignored by the component; it is used by the note allocator when deciding which music device to use.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Synthesizers” (V–2920)

MusicGenericConfigure

Informs the generic music component what services your music component requires and points to any **resources** that are necessary.

```
ComponentResult MusicGenericConfigure (
    MusicComponent    mc,
    long               mode,
    long               flags,
    long               baseResID );
```

mc

The instance of the generic music component. Your software obtains this reference when calling the Component Manager's `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131) function.

mode

Must be 0.

flags

Flags (see below) that control the importation of MIDI files.

baseResID

The **resource** ID of the lowest-numbered resource used by your music component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`kGenericMusicDoMIDI`

Implement normal MIDI messages for note, controllers, and program changes 0–127.

`kGenericMusicBank0...kGenericMusicBank32`

If `kGenericMusicBank0` is set, then bank changes for instruments numbered above 127 will be sent on controller zero; if `kGenericMusicBank32`, then on controller 32. If both flags are set, then the bank is sent on controller 0, and then a 0 value is sent to controller 32.

`kGenericMusicErsatzMIDI`

Some musical devices may internally be driven by a MIDI stream but should not appear to the user to be an external MIDI device. The `kGenericMusicErsatzMIDI` flag instructs the generic music component to allocate channels appropriately and construct MIDI packets. The MIDI packets are always sent to the routine `MusicDerivedMIDISend` (II–975), and never to an external MIDI port.

`kGenericMusicCallKnobs`

Specifies that your music component should receive calls to its routine `MusicDerivedSetKnob` (II–976) for changes to global or part knobs. This flag should be set if your component implements any knobs.

`kGenericMusicCallParts`

Specifies that your music component should receive calls to its routine `MusicDerivedSetPart` (II–979), in order to alter a specific part's polyphony or, in the case of a MIDI device, MIDI channel number.



MusicGenericGetKnobList

kGenericMusicCallInstrument

Specifies that your music component should receive calls to its routine `MusicDerivedSetInstrument` (II-976), in order to set a part to a new instrument. This is for devices that support complete user-instruments with knob lists. If this flag is not set, then the generic music component calls your music component many times to set the value of each knob in the instrument.

kGenericMusicCallNumber

Directs the generic music component to call your music component's `MusicDerivedSetPartInstrumentNumber` (II-979) function, rather than sending standard MIDI program-change and bank-change messages.

kGenericMusicCallROMInstrument

Allows instruments that appear to the user as instruments built into the synthesizer to be stored in the derived component's **resource** file, as 'ROMi' resources. The derived component gets a call to `MusicDerivedSetInstrument` (II-976) when one of these instruments is requested.

Discussion

The `baseResID` parameter is the lowest **resource** ID used by your component for the standard resources described above. Since the resource numbers are relative to this, you can include several music components in a single system extension.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing the Generic Music Component” (V-2925)

MusicGenericGetKnobList

Gets a list of the knobs of a given type for the generic music component.

```
ComponentResult MusicGenericGetKnobList (
    MusicComponent          mc,
    long                    knobType,
    GenericKnobDescriptionListHandle *gkd1H );
```

mc

The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131).

knobType

A constant (see below) that defines the type of knob.

gkd1H

On return, a pointer to a handle to a GenericKnobDescriptionList (IV–2268) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

knobType Constants

kGenericMusicKnob

Value is 1.

kGenericMusicInstrumentKnob

Value is 2.

kGenericMusicDrumKnob

Value is 3.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

MusicGenericGetPart

Gets a part used by the generic music component.

```
ComponentResult MusicGenericGetPart (
    MusicComponent    mc,
    long               partNumber,
    GCPart             **part );
```

mc

The instance of the generic music component. Your software obtains this reference when calling OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

partNumber

The number of the part for this operation.

part

A handle to a GCPart (IV–2263) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

MusicGenericSetResourceNumbers

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

MusicGenericSetResourceNumbers

Undocumented

```
ComponentResult MusicGenericSetResourceNumbers (
    MusicComponent    mc,
    Handle             resourceIDH );
```

mc

The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

resourceIDH

A handle to a **resource** ID.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

MusicGetDescription

Returns a structure describing the synthesizer controlled by the music component device.

```
ComponentResult MusicGetDescription (
    MusicComponent    mc,
    SynthesizerDescription    *sd );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

`sd`

Pointer to a `SynthesizerDescription` (IV–2465) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Synthesizers” (V–2920)

MusicGetDeviceConnection

Determines how many hardware synthesizers are available to a music component and gets the IDs for those devices.

```
ComponentResult MusicGetDeviceConnection (
    MusicComponent mc,
    long index,
    long *id1,
    long *id2 );
```

`mc`

Music component returned by `NAGetRegisteredMusicDevice` (II–1028).

`index`

Index of the device for which you want to find out the IDs. Set to 0 if you are calling to get the number of hardware devices.

`id1`

On return, a hardware synthesizer ID.

`id2`

On return, another hardware synthesizer ID.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

To get the number of hardware synthesizers available to the music component specified in the `mc` parameter and an index you can use to request ID numbers for a specific device, call this function with a value of 0 for the `index` parameter. You can then pass an index value in the `index` parameter, and the function returns hardware synthesizer IDs in the `id1` and `id2` parameters.

MusicGetDrumKnobDescription

Special Considerations

This function is implemented only for hardware synthesizers, such as PCI card devices.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Synthesizers” (V–2920)

MusicGetDrumKnobDescription

Returns a description of a drum kit knob.

```
ComponentResult MusicGetDrumKnobDescription (
    MusicComponent      mc,
    long                knobIndex,
    KnobDescription      *mkd );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

knobIndex

A knob index or knob ID.

mkd

A pointer to a `KnobDescription` (IV–2299) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Synthesizers” (V–2920)

MusicGetDrumKnobDescriptionObsolete

Undocumented

```
ComponentResult MusicGetDrumKnobDescriptionObsolete (
    MusicComponent      mc,
```

```
long          knobIndex,
void          *mkd );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

knobIndex

A knob index or knob ID.

mkd

A pointer to a `KnobDescription` (IV–2299) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

MusicGetDrumNames

Undocumented

```
ComponentResult MusicGetDrumNames (
    MusicComponent mc,
    long           modifiableInstruments,
    Handle         *instrumentNumbers,
    Handle         *instrumentNames );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

modifiableInstruments

Undocumented

instrumentNumbers

Undocumented

instrumentNames

A pointer to a handle to the requested list of instrument name strings, formatted as a short integer followed by packed strings.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

MusicGetInfoText

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

MusicGetInfoText

Undocumented

```
ComponentResult MusicGetInfoText (
    MusicComponent mc,
    long selector,
    Handle *textH,
    Handle *styleH );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II-1028).

selector

Undocumented

textH

Undocumented

styleH

Undocumented

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

MusicGetInstrumentAboutInfo

Obtains the information about an instrument that appears in its About box.

```
ComponentResult MusicGetInstrumentAboutInfo (
    MusicComponent mc,
    long part,
    InstrumentAboutInfo *iai );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

part

Number of the part containing the instrument for which you want information.

iai

On return, a pointer to an `InstrumentAboutInfo` (IV–2293) structure for the instrument currently on the specified synthesizer part.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Instruments and Parts” (V–2921)

MusicGetInstrumentInfo

Obtains a list of instruments supported by a synthesizer.

```
ComponentResult MusicGetInstrumentInfo (
    MusicComponent      mc,
    long                getInstrumentInfoFlags,
    InstrumentInfoListHandle *infoListH );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

getInstrumentInfoFlags

Flags (see below) that specify limits to the list of instruments.

infoListH

On return, a pointer to a handle to an `InstrumentInfoList` (IV–2294) structure that contains the list of instruments. This handle must be disposed of by the caller.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

getInstrumentInfoFlags Constants

`kGetInstrumentInfoNoBuiltIn`

Don’t return built-in instruments.

MusicGetInstrumentKnobDescription

kGetInstrumentInfoMidiUserInst

Do return user instruments for a MIDI device.

kGetInstrumentInfoNoIText

Don't return international text strings.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Instruments and Parts” (V–2921)

MusicGetInstrumentKnobDescription

Obtains the description of an instrument knob.

```
ComponentResult MusicGetInstrumentKnobDescription (
    MusicComponent      mc,
    long                knobIndex,
    KnobDescription      *mkd );
```

mc

Music component instance identifier returned by NAGetRegisteredMusicDevice (II–1028).

knobIndex

A knob index or knob ID.

mkd

On return, a KnobDescription (IV–2299) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Synthesizers” (V–2920)

MusicGetInstrumentKnobDescriptionObsolete

Undocumented

```
ComponentResult MusicGetInstrumentKnobDescriptionObsolete (
    MusicComponent    mc,
    long              knobIndex,
    void              *mkd );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

knobIndex

A knob index or knob ID.

mkd

On return, a `KnobDescription` (IV–2299) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

MusicGetInstrumentNames

Undocumented

```
ComponentResult MusicGetInstrumentNames (
    MusicComponent    mc,
    long              modifiableInstruments,
    Handle             *instrumentNames,
    Handle             *instrumentCategoryLasts,
    Handle             *instrumentCategoryNames );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

modifiableInstruments

Undocumented

instrumentNames

A pointer to a handle to the requested list of instrument name strings, formatted as a short integer followed by packed strings.

MusicGetKnob

`instrumentCategoryLasts`

A pointer to a handle to a group of short integers, the first of which contains the number of integers to follow.

`instrumentCategoryNames`

A pointer to a handle to the requested list of instrument category name strings, formatted as a short integer followed by packed strings.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

MusicGetKnob

Returns the value of the specified global synthesizer knob.

```
ComponentResult MusicGetKnob (  
    MusicComponent    mc,  
    long              knobID );
```

`mc`

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

`knobID`

Knob index or ID.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

A global knob controls an aspect of the entire synthesizer. It is not specific to a part within the synthesizer.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Synthesizers” (V–2920)

See Also

For the corresponding set function, see `MusicSetKnob` (II–1007).

MusicGetKnobDescription

Returns a pointer to an initialized knob description structure describing a global synthesizer knob.

```
ComponentResult MusicGetKnobDescription (
    MusicComponent    mc,
    long               knobIndex,
    KnobDescription    *mkd );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

knobIndex

Knob index or ID.

mkd

Pointer to a `KnobDescription` (IV–2299) structure. The initialized structure provides default values associated with the particular knob.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

A global knob controls an aspect of the entire synthesizer; it is not limited to a part within the synthesizer. You can use the information returned by a call to this function to reset a knob to some known, usable value.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Synthesizers” (V–2920)

MusicGetKnobDescriptionObsolete

Undocumented

```
ComponentResult MusicGetKnobDescriptionObsolete (
    MusicComponent    mc,
    long               knobIndex,
    void               *mkd );
```

MusicGetKnobSettingStrings

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

knobIndex

Knob index or ID.

mkd

Pointer to a `KnobDescription` (IV–2299) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

MusicGetKnobSettingStrings

Returns a list of knob setting names known by the specified music component.

```
ComponentResult MusicGetKnobSettingStrings (
    MusicComponent mc,
    long knobIndex,
    long isGlobal,
    Handle *settingsNames,
    Handle *settingsCategoryLasts,
    Handle *settingsCategoryNames );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

knobIndex

The knob index or knob ID.

isGlobal

If a knob index is used, indicates whether the specified knob is a global knob.

settingsNames

The requested list of knob setting strings formatted as a short followed by packed strings.

settingsCategoryLasts

A group of short integers, the first of which contains the number of shorts to follow.

settingsCategoryNames

Knob setting category names formatted as a short integer followed by a list of names.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

All handles must be disposed of by the caller.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Synthesizers” (V–2920)



MusicGetMasterTune

Returns the synthesizer’s master tuning as a fixed-point value in semitones.

```
ComponentResult MusicGetMasterTune (
    MusicComponent mc );
```

mc

Music component instance identifier returned by NAGetRegisteredMusicDevice (II–1028).

function result A fixed-point value representing the synthesizer’s master tuning. The value is a fixed 16.16 number, allowing shifts by fractional values.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Miscellaneous Music Component Functions” (V–2922)

See Also

For the corresponding set function, see MusicSetMasterTune (II–1008).

MusicGetMIDIPorts

Returns the number of input and output ports a MIDI device has.

MusicGetMIDIProc

```
ComponentResult MusicGetMIDIPorts (
    MusicComponent mc,
    long *inputPortCount,
    long *outputPortCount );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

inputPortCount

On return, the number of input MIDI ports available to the music component.

outputPortCount

On return, the number of output MIDI ports available to the music component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

This call is implemented only for hardware synthesizers, such as PCI card devices.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Synthesizers” (V–2920)

MusicGetMIDIProc

Returns a pointer to the procedure a music component is using to process external MIDI notes.

```
ComponentResult MusicGetMIDIProc (
    MusicComponent mc,
    MusicMIDISendUPP *midiSendProc,
    long *refCon );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

midiSendProc

Pointer to a MIDI serial port `MusicMIDISendProc` (III–2110) callback that processes external MIDI notes. This function was set by a previous call to `MusicSetMIDIProc` (II–1009). If no function has been set with `MusicSetMIDIProc`, this parameter returns 0.

refCon

A reference constant. The Movie Toolbox passes this reference constant to your MusicMIDISendProc each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Synthesizers” (V–2920)

See Also

For the corresponding set function, see MusicSetMIDIProc (II–1009).



MusicGetPart

Returns the MIDI channel and maximum polyphony for a particular part.

```
ComponentResult MusicGetPart (
    MusicComponent    mc,
    long              part,
    long              *midiChannel,
    long              *polyphony );
```

mc

Music component instance identifier returned by NAGetRegisteredMusicDevice (II–1028).

part

The music component part requested.

midiChannel

On return, a pointer to a MIDI channel. For non-MIDI devices, the MIDI channel pointed to by this parameter is 0.

polyphony

On return, a pointer to the maximum number of voices or polyphony for the part..

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

MusicGetPartAtomicInstrument

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding set function, see `MusicSetPart` (II–1010).

MusicGetPartAtomicInstrument

Returns the atomic instrument currently in a part.

```
ComponentResult MusicGetPartAtomicInstrument (  
    MusicComponent      mc,  
    long                part,  
    AtomicInstrument    *ai,  
    long                flags );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

part

The part with the atomic instrument.

ai

On return, an atomic instrument.

flags

A constant (see below) that specifies what pieces of information about an atomic instrument the caller is interested in.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`kGetAtomicInstNoExpandedSamples`

Eliminate the expanded samples.

`kGetAtomicInstNoOriginalSamples`

Eliminate the original samples.

`kGetAtomicInstNoSamples`

Eliminate both the original and expanded samples.

`kGetAtomicInstNoKnobList`

Eliminate the knob list.

`kGetAtomicInstNoInstrumentInfo`

Eliminate the About box information.

kGetAtomicInstOriginalKnobList
Include the original knob list.

kGetAtomicInstAllKnobs
Include the current knob list.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`
Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding set function, see `MusicSetPartAtomicInstrument` (II–1011).



MusicGetPartController

Returns the value of a specified controller on a specified part.

```
ComponentResult MusicGetPartController (
    MusicComponent    mc,
    long              part,
    MusicController    controllerNumber );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

part

Part whose controller value you want to get.

controllerNumber

On return, the controller number; see “Music Controllers” (IV–2682).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`
Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding set function, see `MusicSetPartController` (II–1012).

MusicGetPartInstrumentNumber

MusicGetPartInstrumentNumber

Returns the instrument number currently assigned to a part.

```
ComponentResult MusicGetPartInstrumentNumber (  
    MusicComponent    mc,  
    long              part );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

part

Part number containing the instrument.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding set function, see `MusicSetPartInstrumentNumber` (II–1013).

MusicGetPartKnob

Retrieves the current value of a knob for a part.

```
ComponentResult MusicGetPartKnob (  
    MusicComponent    mc,  
    long              part,  
    long              knobID );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

part

The part number.

knobID

The knob index or ID.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding set function, see MusicSetPartKnob (II–1014).

MusicGetPartName

Returns the string name of a part.

```
ComponentResult MusicGetPartName (
    MusicComponent mc,
    long part,
    StringPtr name );
```

mc

Music component instance identifier returned by NAGetRegisteredMusicDevice (II–1028).

part

Part to get the name of.

name

On return, a string containing the part name.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The name string is used by selection dialog boxes or configuration information.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding set function, see MusicSetPartName (II–1015).

MusicMediaGetIndexedTunePlayer

MusicMediaGetIndexedTunePlayer

Undocumented

```
ComponentResult MusicMediaGetIndexedTunePlayer (
    ComponentInstance    ti,
    long                  sampleDescIndex,
    ComponentInstance     *tp );
```

ti

Undocumented

sampleDescIndex

Undocumented

tp

A pointer to a tune player component instance.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Related Java Methods

```
quicktime.std.movies.media.MusicMediaHandler.getIndexedTunePlayer(),
quicktime.std.music.TunePlayer.fromMusicMediaHandler()
```

MusicPlayNote

Plays a note on a specified part at a specified pitch and velocity.

```
ComponentResult MusicPlayNote (
    MusicComponent    mc,
    long               part,
    long               pitch,
    long               velocity );
```

mc

Music component instance identifier returned by NAGetRegisteredMusicDevice (II–1028).

part

The part to play the note on.

pitch

The pitch at which to play the note. If the pitch is specified by a number from 0 to 127, it is a MIDI pitch, where 60 is middle C. If the pitch is a positive number above 65535, the value is a fixed-point pitch value. Thus, microtonal values may be specified.

velocity

How hard to strike the key. Values are 0–127 where 0 is silence. Velocity refers to how hard the key is struck (if performed on a keyboard-instrument); typically, this translates directly to volume, but on many synthesizers this also subtly alters the timbre of the tone.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The current note continues to play until a MusicPlayNote function with the same pitch and velocity of 0 turns the note off.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Synthesizers” (V–2920)

MusicReceiveMIDI

Undocumented

```
ComponentResult MusicReceiveMIDI (
    MusicComponent      mc,
    MusicMIDIReadHookUPP readHook,
    long                refCon );
```

mc

Music component instance identifier returned by NAGetRegisteredMusicDevice (II–1028).

readHook

A **Universal Procedure Pointer** that accesses a MusicMIDIReadHookProc (III–2109) callback.

MusicResetPart

refCon

A reference constant to be passed to your callback. Use this constant to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

MusicResetPart

Silences all sounds on a specified part and resets all controllers on that part to their default values.

```
ComponentResult MusicResetPart (  
    MusicComponent mc,  
    long part );
```

mc

Music component instance identifier returned by NAGetRegisteredMusicDevice (II–1028).

part

The number of the part.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The default value to which controllers on the part are set is 0 for all controllers except volume. Volume is set to its maximum, 32767 (0x7FFF).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Instruments and Parts” (V–2921)

MusicSendMIDI

Sends a MIDI packet to a specified port.

```
ComponentResult MusicSendMIDI (
    MusicComponent    mc,
    long              portIndex,
    MusicMIDIPacket    *mp );
```

mc

Music component instance returned by `NAGetRegisteredMusicDevice` (II–1028).

portIndex

The index of the port to send the MIDI packet to. The index value is 1 through the port count returned by `MusicGetMIDIPorts` (II–997).

mp

A pointer to the music MIDI packet to be sent. The function sends the MIDI music packet specified by this parameter to the specified port.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

This call is implemented only for hardware synthesizers, such as PCI card devices.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Synthesizers” (V–2920)

MusicSetKnob

Modifies the value of the specified global synthesizer knob.

```
ComponentResult MusicSetKnob (
    MusicComponent    mc,
    long              knobID,
    long              knobValue );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

knobID

Knob index or ID.

knobValue

Value for specified knob.

MusicSetMasterTune

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

A global knob controls an aspect of the entire synthesizer; it is not limited to a part within the synthesizer.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Synthesizers” (V–2920)

See Also

For the corresponding get function, see MusicGetKnob (II–994).

MusicSetMasterTune

Alters a synthesizer’s master tuning.

```
ComponentResult MusicSetMasterTune (  
    MusicComponent mc,  
    long masterTune );
```

mc

Music component instance identifier returned by NAGetRegisteredMusicDevice (II–1028).

masterTune

The amount by which to transpose the entire synthesizer in pitch. The value is a fixed 16.16 number, allowing shifts by fractional values.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Miscellaneous Music Component Functions” (V–2922)

See Also

For the corresponding get function, see MusicGetMasterTune (II–997).

MusicSetMIDIProc

Notifies the music component what procedure to call when it needs to send MIDI data.

```
ComponentResult MusicSetMIDIProc (
    MusicComponent      mc,
    MusicMIDISendUPP    midiSendProc,
    long                refCon );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

midiSendProc

A pointer to the `MusicMIDISendProc` (III–2110) callback to use when sending MIDI data.

refCon

A reference constant value. The Movie Toolbox passes this reference constant to your callback each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This call is implemented only by music components for MIDI synthesizers.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Synthesizers” (V–2920)

See Also

For the corresponding get function, see `MusicGetMIDIProc` (II–998).

MusicSetOfflineTimeTo

Advances the synthesizer clock when the synthesizer is not running in real time.

```
ComponentResult MusicSetOfflineTimeTo (
    MusicComponent      mc,
    long                newTimeStamp );
```

MusicSetPart

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

newTimeStamp

The number of samples to synthesize.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The synthesizer may not be running in real time due to a call to `MusicStartOffline` (II–1016). Setting the time generates audio output from the synthesizer.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Miscellaneous Music Component Functions” (V–2922)

MusicSetPart

Sets the MIDI channel and maximum polyphony for a specified part.

```
ComponentResult MusicSetPart (  
    MusicComponent    mc,  
    long               part,  
    long               midiChannel,  
    long               polyphony );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

part

Part whose MIDI channel and polyphony are to be set.

midiChannel

The MIDI channel to set the part to. For non-MIDI devices, set this parameter to 0.

polyphony

The maximum number of voices or polyphony for the part.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding get function, see `MusicGetPart` (II–999).

MusicSetPartAtomicInstrument

Initializes a part with an atomic instrument.

```
ComponentResult MusicSetPartAtomicInstrument (
    MusicComponent      mc,
    long                part,
    AtomicInstrumentPtr  aiP,
    long                flags );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

part

The part to initialize with the atomic instrument to.

aiP

The atomic instrument.

flags

Constants (see below) that specify details of initializing a part with an atomic instrument.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`kGetAtomicInstNoExpandedSamples`

Eliminate the expanded samples.

`kGetAtomicInstNoOriginalSamples`

Eliminate the original samples.

`kGetAtomicInstNoSamples`

Eliminate both the original and expanded samples.

`kGetAtomicInstNoKnobList`

Eliminate the knob list.

MusicSetPartController

kGetAtomicInstNoInstrumentInfo

Eliminate the About box information.

kGetAtomicInstOriginalKnobList

Include the original knob list.

kGetAtomicInstAllKnobs

Include the current knob list.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding get function, see MusicGetPartAtomicInstrument (II–1000).

MusicSetPartController

Initializes the value of a specified controller on a specified part.

```
ComponentResult MusicSetPartController (
    MusicComponent      mc,
    long                part,
    MusicController      controllerNumber,
    long                controllerValue );
```

mc

Music component instance identifier returned by NAGetRegisteredMusicDevice (II–1028).

part

Part whose controller value you want to set.

controllerNumber

Controller number; see “Music Controllers” (IV–2682).

controllerValue

Value for controller.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding get function, see `MusicGetPartController` (II–1001).

MusicSetPartInstrumentNumber

Superseded by `MusicSetPartInstrumentNumberInterruptSafe` (II–1013).

```
ComponentResult MusicSetPartInstrumentNumber (
    MusicComponent mc,
    long part,
    long instrumentNumber );
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Instruments and Parts” (V–2921)

MusicSetPartInstrumentNumberInterruptSafe

Initializes a part with a particular instrument.

```
ComponentResult MusicSetPartInstrumentNumberInterruptSafe (
    MusicComponent mc,
    long part,
    long instrumentNumber );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

part

Part to be initialized.

instrumentNumber

Number of instrument to initialize part with. You can use `MusicFindTone` (II–981) to get an instrument number.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

MusicSetPartKnob

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding get function, see `MusicGetPartInstrumentNumber` (II–1002).

MusicSetPartKnob

Sets a knob for a specified part.

```
ComponentResult MusicSetPartKnob (
    MusicComponent mc,
    long part,
    long knobID,
    long knobValue );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

part

The part number.

knobID

The index or ID of the knob to be set.

knobValue

The value to set the knob to.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding get function, see `MusicGetPartKnob` (II–1002).

MusicSetPartName

Changes the name of an instrument in a specified part.

```
ComponentResult MusicSetPartName (
    MusicComponent    mc,
    long              part,
    StringPtr         name );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

part

Part to apply name to.

name

A pointer to the name to apply to part.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You might want to change the name of a modified instrument before saving it. The instrument name string is used by selection dialog and configuration information boxes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Instruments and Parts” (V–2921)

See Also

For the corresponding get function, see `MusicGetPartName` (II–1003).

MusicSetPartSoundLocalization

Passes sound localization data to a specified synthesizer part.

```
ComponentResult MusicSetPartSoundLocalization (
    MusicComponent    mc,
    long              part,
    Handle             data );
```

MusicStartOffline

mc

Music component instance identifier.

part

The part to pass the data to.

data

The sound localization data.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Use the functions described in this section to get and modify the master tuning of the synthesizer, to play off line, and to allow the music component to perform tasks it must perform at foreground task time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Managing Instruments and Parts” (V–2921)

MusicStartOffline

Informs the QuickTime music synthesizer that the music will not be played through the speakers.

```
ComponentResult MusicStartOffline (
    MusicComponent      mc,
    unsigned long        *numChannels,
    unsignedFixed        *sampleRate,
    unsigned short       *sampleSize,
    MusicOfflineDataUPP  dataProc,
    long                 dataProcRefCon );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

numChannels

Number of channels in the music sample; 1 indicates monaural, 2 indicates stereo.

sampleRate

The number of samples per second.

sampleSize

The size of the music sample: 8-bit or 16-bit.

dataProc

A pointer to a MusicOfflineDataProc (III–2110) callback to handle the audio data.

dataProcRefCon

A reference constant to pass to the MusicOfflineDataProc callback. Use this parameter to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Audio data will be sent to a function that will create a sound file to be played back later. You pass this function the requested values for the numChannels, sampleRate, and sampleSize parameters. When the function returns, those parameters contain the actual values used.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Miscellaneous Music Component Functions” (V–2922)

MusicStorePartInstrument

Puts whatever instrument is on the specified part into the synthesizer’s instrument store.

```
ComponentResult MusicStorePartInstrument (
    MusicComponent mc,
    long part,
    long instrumentNumber );
```

mc

Music component instance identifier returned by NAGetRegisteredMusicDevice (II–1028).

part

Part containing the instrument to be stored.

MusicTask

`instrumentNumber`

Instrument number at which to store the part. The value must be between 1 and the synthesizer's modifiable instrument count, as defined by the `modifiableInstrumentCount` field of the synthesizer's `SynthesizerDescription` (IV–2465) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function lets you store modified instruments.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Instruments and Parts” (V–2921)

MusicTask

Allows a music component to perform tasks it must perform at foreground task time.

```
ComponentResult MusicTask (  
    MusicComponent    mc );
```

`mc`

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II–1028).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function must be called periodically. In the case of the QuickTime music synthesizer, instruments cannot be loaded from disk at interrupt time, so if the `NASetInstrumentNumberInterruptSafe` (II–1044) function is called, the instrument is loaded during the next `MusicTask` call.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Miscellaneous Music Component Functions” (V–2922)

MusicUseDeviceConnection

Tells a music component which hardware synthesizer to talk to.

```
ComponentResult MusicUseDeviceConnection (
    MusicComponent mc,
    long id1,
    long id2 );
```

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (II-1028).

id1

The ID of the device returned in the `id1` parameter of `MusicGetDeviceConnection` (II-987).

id2

The ID of the device returned in the `id2` parameter of `MusicGetDeviceConnection` (II-987).

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Special Considerations

This call is implemented only for hardware synthesizers, such as PCI card devices.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Managing Synthesizers” (V-2920)

N

NACopyrightDialog

Displays a copyright dialog box with information specific to a music device.

```
ComponentResult NACopyrightDialog (
    NoteAllocator na,
    PicHandle p,
    StringPtr author,
```

NADisposeNoteChannel

```
StringPtr    copyright,  
StringPtr    other,  
StringPtr    title,  
ModalFilterUPP filterProc,  
long         refCon );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

p

A handle to a `Picture` (IV–2331) structure containing the image **resource** for the dialog box.

author

A pointer to a string containing author information.

copyright

A pointer to a string containing copyright information.

other

A pointer to a string containing any additional information.

title

A pointer to a string containing title information.

filterProc

Pointer to a `ModalFilterProc` (III–2098) callback.

refCon

A reference constant value. The Movie Toolbox passes this reference constant to your `ModalFilterProc` each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Interface Tools” (V–2918)

Related Java Methods

`quicktime.std.music.NoteAllocator.copyrightDialog()`

NADisposeNoteChannel

Deletes a specified note channel.

```
ComponentResult NADisposeNoteChannel (
    NoteAllocator    na,
    NoteChannel      noteChannel );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

noteChannel

Note channel to be disposed. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

N

NAFindNoteChannelTone

Locates the instrument that best fits a requested tone description for a specific channel.

```
ComponentResult NAFindNoteChannelTone (
    NoteAllocator    na,
    NoteChannel      noteChannel,
    ToneDescription  *td,
    long             *instrumentNumber );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

noteChannel

The note channel for which you want an instrument. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

td

A `ToneDescription` (IV–2487) structure that describes the instrument fit.

instrumentNumber

On return, the number of the instrument that best fits the tone description.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

NAGetController

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.findTone()`

NAGetController

Retrieves the controller settings for a note channel.

```
ComponentResult NAGetController (
    NoteAllocator    na,
    NoteChannel      noteChannel,
    long             controllerNumber,
    long             *controllerValue );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

noteChannel

Note channel for which to get controller settings. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

controllerNumber

The controller for which to get settings; see “Music Controllers” (IV–2682).

controllerValue

On return, the value for the controller setting, typically 0 (0x00.00) to 32767 (0x7F.FF).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.getController()`

See Also

For the corresponding set function, see `NASetController` (II–1042).

NAGetDefaultMIDIInput

Obtains external MIDI connection information.

```
ComponentResult NAGetDefaultMIDIInput (
    NoteAllocator      na,
    SynthesizerConnections *sc );
```

na

You obtain the note allocator identifier from `OpenComponent` (II–1130).

sc

On return, an initialized `SynthesizerConnections` (IV–2464) structure containing information about the external MIDI device attached to the system that has been selected as the default MIDI input device.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This routine calls the QuickTime MIDI components to query them. The external MIDI device provides note input directly to the note allocator.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Configuration and Utilities” (V–2919)

Related Java Methods

`quicktime.std.music.NoteAllocator.getDefaultMIDIInput()`

See Also

For the corresponding set function, see `NASetDefaultMIDIInput` (II–1043).

NAGetIndNoteChannel

Returns the number of note channels handled by the specified note allocator instance.

```
ComponentResult NAGetIndNoteChannel (
    NoteAllocator na,
```

NAGetKnob

```
long          index,  
NoteChannel   *nc,  
long          *seed );
```

na

You obtain the note allocator identifier from the Component Manager's `OpenComponent` (II-1130) function.

index

The index of the note channel. If 0, the result is still the number of note channels, but the *nc* parameter is not filled out.

nc

The note channel requested.

seed

A number that changes on successive calls if anything significant changes about a note channel; for example, if the note channel has been reallocated or released.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This function can also return a requested note channel. To get a count of the note channels, pass 0 in the *index* parameter. To get a specific note channel, pass the index value returned by a previous call to `NAGetIndNoteChannel`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V-2916)

Related Java Methods

```
quicktime.std.music.NoteAllocator.numNoteChannels(),  
quicktime.std.music.NoteAllocator.getIndNoteChannel()
```

NAGetKnob

Obtains the value of a knob for a given note channel.

```
ComponentResult NAGetKnob (  
    NoteAllocator   na,  
    NoteChannel     noteChannel,  
    long            knobNumber,  
    long            *knobValue );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

noteChannel

The note channel whose knob value you want to get. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

knobNumber

The index or ID of the knob whose value you want to get.

knobValue

On return, a pointer to the value of the knob.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.getKnob()`

See Also

For the corresponding set function, see `NASetKnob` (II–1045).

NAGetMIDIPorts

The MIDI input and output ports available to a note allocator.

```
ComponentResult NAGetMIDIPorts (
    NoteAllocator      na,
    QTMIDIPortListHandle *inputPorts,
    QTMIDIPortListHandle *outputPorts );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

inputPorts

On return, a handle giving the number of input ports (the first two bytes) followed by a list of `QTMIDIPort` (IV–2357) structures.

NAGetNoteChannelInfo

outputPorts

On return, a handle giving the number of output ports (the first two bytes) followed by a list of QTMIDIPort (IV–2357) structures.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This routine calls the QuickTime MIDI components to query them.

Special Considerations

NAGetMIDIPorts is the correct call for applications to make. They should not call QTMIDIGetMIDIPorts.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Note Allocator Configuration and Utilities” (V–2919)

Related Java Methods

quicktime.std.music.NoteAllocator.getMIDIInPorts(),
quicktime.std.music.NoteAllocator.getMIDIOutPorts()

NAGetNoteChannelInfo

Returns the index of the music component for the allocated channel and its part number on that music component.

```
ComponentResult NAGetNoteChannelInfo (
    NoteAllocator    na,
    NoteChannel      noteChannel,
    long             *index,
    long             *part );
```

na

You obtain the note allocator identifier by calling OpenComponent (II–1130).

noteChannel

Note channel to get information about. You obtain the note channel identifier from NANewNoteChannel (II–1030) or NANewNoteChannelFromAtomicInstrument (II–1031).

index

Music component index.

part

Music component part pointer.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The NAGetNoteChannelInfo function allows direct access to the music component allocated to the note channel by the note allocator. The index returned becomes invalid if music components are subsequently registered or unregistered.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

quicktime.std.music.NoteChannel.getIndexInfo(),
quicktime.std.music.NoteChannel.getPartInfo()

N

NAGetNoteRequest

Retrieves the NoteRequest (IV–2323) structure that was passed to a note channel.

```
ComponentResult NAGetNoteRequest (
    NoteAllocator    na,
    NoteChannel      noteChannel,
    NoteRequest      *nrOut );
```

na

You obtain the note allocator identifier from the Component Manager’s OpenComponent (II–1130) function.

noteChannel

The note channel whose note request you want to get. You obtain the note channel identifier from NANewNoteChannel (II–1030) or NANewNoteChannelFromAtomicInstrument (II–1031).

nrOut

On return, the NoteRequest (IV–2323) structure that was used when the specified note channel was allocated.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

NAGetRegisteredMusicDevice

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.getNoteRequest()`

NAGetRegisteredMusicDevice

Returns details about music components registered to the specified note allocator instance.

```
ComponentResult NAGetRegisteredMusicDevice (
    NoteAllocator      na,
    long               index,
    OSType             *synthType,
    Str31              name,
    SynthesizerConnections *connections,
    MusicComponent     *mc );
```

na

You obtain the note allocator identifier from the Component Manager’s `OpenComponent` (II–1130) function.

index

The index of the music component to get information about. To get a count of the registered music components, pass 0 in the `index` parameter. The return value is the count of components. To get information about one of the music components registered with the note allocator, pass the music component index in the `index` parameter. The index value can be 1 through the number of registered components returned by a previous call to `NAGetRegisteredMusicDevice`.

synthType

Synthesizer type.

name

Synthesizer name as a text string.

connections

A synthesizer connections for MIDI devices structure.

mc

Music component instance identifier.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.**Discussion**

If you request information about a specific registered music component, this function returns the type of synthesizer the component supports in the `synthType` parameter, the name of the synthesizer in the `name` parameter, and the music component identifier in the `mc` parameter. For MIDI devices, it returns a pointer to a MIDI devices structure with information about the synthesizer connections.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming InfoC interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Configuration and Utilities” (V–2919)

Related Java Methods

```
quicktime.std.music.NoteAllocator.numMusicComponents(),
quicktime.std.music.NoteAllocator.getRegisteredMusicDevice()
```

N**NALoseDefaultMIDIInput**

Removes the external default MIDI service procedure call, if previously defined by `NAUseDefaultMIDIInput` (II–1052).

```
ComponentResult NALoseDefaultMIDIInput (
    NoteAllocator    na );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).*function result* See “Error Codes” (IV–2663). Returns noErr if there is no error.**Version Notes**

Introduced in QuickTime 3 or earlier.

Programming InfoC interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

```
quicktime.std.music.NoteAllocator.loseDefaultMIDIInput()
```

NANewNoteChannel

NANewNoteChannel

Requests a new note channel with the qualities described in a `NoteRequest` (IV–2323) structure.

```
ComponentResult NANewNoteChannel (  
    NoteAllocator    na,  
    NoteRequest      *noteRequest,  
    NoteChannel      *outChannel );
```

`na`

You obtain the note allocator identifier from the Component Manager’s `OpenComponent` (II–1130) function.

`noteRequest`

A pointer to a `NoteRequest` (IV–2323) structure.

`outChannel`

On return, a pointer to an identifier for a new note channel or `NIL` if the function fails to create a note channel.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function searches all available music components for the instrument that best matches the specifications in the `ToneDescription` (IV–2487) structure that is contained within the `noteRequest` parameter. If an error occurs, the new note channel is initialized to `NIL`. The caller can request an instrument that is not currently allocated to a part. In that case, this function may return a value in `outChannel`, even though the request cannot initially be satisfied. The note channel may become valid at a later time, as other note channels are released or other music components are registered.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.NoteChannel()`

NANewNoteChannelFromAtomicInstrument

Requests a new note channel for an atomic instrument.

```
ComponentResult NAnewNoteChannelFromAtomicInstrument (
    NoteAllocator      na,
    AtomicInstrumentPtr instrument,
    long               flags,
    NoteChannel        *outChannel );
```

na

You obtain the note allocator identifier from the Component Manager's `OpenComponent` (II-1130) function.

instrument

A pointer to the atomic instrument. This may be a dereferenced locked QT atom container.

flags

Flags (see below) that specify details of initializing a part with an atomic instrument.

outChannel

On return, a pointer to an identifier for a new note channel or `NIL` if the function fails to create a note channel.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

flags Constants

`kSetAtomicInstKeepOriginalInstrument`

Keep original sample after expansion.

`kSetAtomicInstShareAcrossParts`

Remove the instrument when the application quits.

`kSetAtomicInstCallerTosses`

The caller isn't keeping a copy of the atomic instrument for later calls to `NASetAtomicInstrument` (II-1041).

`kSetAtomicInstDontPreprocess`

Don't expand the sample. You would only set this bit if you know the instrument is digitally clean or you got it from a `MusicGetPartAtomicInstrument` (II-1000) call.

Discussion

This function takes a note allocator identifier in the *na* parameter and a pointer to the atomic instrument you are requesting a new channel for in the *instrument* parameter. Among other things, you can specify how to handle the expanded

NAPickArrangement

sample with the `flags` parameter. The function returns the note channel allocated for the instrument in the `outChannel` parameter, or `NIL` if an error occurs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.AtomicInstrument.newNoteChannel()`

NAPickArrangement

Displays a dialog box to allow instrument selection.

```
ComponentResult NAPickArrangement (
    NoteAllocator      na,
    ModalFilterUPP     filterProc,
    StringPtr          prompt,
    long               zero1,
    long               zero2,
    Track              t,
    StringPtr          songName );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

filterProc

A **Universal Procedure Pointer** to a `ModalFilterProc` (III–2098) callback.

prompt

A pointer to a dialog box prompt string.

zero1

Must be 0.

zero2

Must be 0.

t

The arrangement movie track number.

songName

A pointer to the name of a song to display in the dialog box.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Interface Tools” (V–2918)

Related Java Methods

`quicktime.std.music.NoteAllocator.pickArrangement()`

NAPickEditInstrument

Presents a user interface for changing the instrument in a live note channel or modifying an atomic instrument.

```
ComponentResult NAPickEditInstrument (
    NoteAllocator      na,
    ModalFilterUPP     filterProc,
    StringPtr          prompt,
    long               refCon,
    NoteChannel        nc,
    AtomicInstrument    ai,
    long               flags );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

filterProc

Pointer to a `ModalFilterProc` (III–2098) callback.

prompt

Dialog box prompt “New Instrument”.

refCon

A reference constant value. The Movie Toolbox passes this reference constant to your `ModalFilterProc` callback each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

nc

The live note channel that appears in the dialog box. If you specify a note channel, set the `ai` parameter to 0. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).



NAPickEditInstrument

a i

The atomic instrument that appears in the dialog box. If you specify an atomic instrument, set the *nc* parameter to 0. You obtain the atomic instrument from `InstrumentGetInst` (II–767).

flags

Flags (see below) that limit the instruments presented. If the `kPickDontMix` flag is set, the dialog box does not display a mix of synthesizer part types. For example, if the current instrument is a drum, only available drums appear in the dialog box. The `kPickSameSynth` flag allows selections only within the current synthesizer. The `kPickUserInsts` flag allows user modifiable instruments to appear. If the `kPickEditAllowPick` flag is not set, no dialog box appears.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`kPickDontMix`

Show either all drum kits or all instruments depending on the current instrument. For example, if it’s a drum kit, show only drum kits.

`kPickSameSynth`

Show only instruments from the current synthesizer.

`kPickUserInsts`

Show modifiable instruments in addition to ROM instruments.

`kPickEditAllowPick`

Present the instrument picker dialog box. Used only with `NAPickEditInstrument`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Interface Tools” (V–2918)

Related Java Methods

`quicktime.std.music.AtomicInstrument.pickEditInstrument()`,

`quicktime.std.music.NoteChannel.pickEditInstrument()`

See Also

See `NAPickInstrument` (II–1035).

NAPickInstrument

Presents a user interface for picking an instrument.

```
ComponentResult NAPickInstrument (
    NoteAllocator      na,
    ModalFilterUPP     filterProc,
    StringPtr          prompt,
    ToneDescription     *sd,
    unsigned long       flags,
    long               refCon,
    long               reserved1,
    long               reserved2 );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

filterProc

Pointer to a `ModalFilterProc` (III–2098) callback.

prompt

A pointer to the dialog box prompt “New Instrument”.

sd

On entry, the tone description of the instrument that appears in the picker dialog box. On return, a tone description of the instrument the user selected.

flags

Flags (see below) that determine whether to display the picker dialog box and what instruments appear for selection. If the `kPickDontMix` flag is set, the dialog box does not display a mix of synthesizer part types. For example, if the current instrument is a drum, only available drums appear in the dialog box. The `kPickSameSynth` flag allows selections only within the current synthesizer. The `kPickUserInsts` flag allows user modifiable instruments to appear. The `kPickEditAllowPick` flag is used only with `NAPickEditInstrument` (II–1033).

refCon

A reference constant value. The Movie Toolbox passes this reference constant to your `ModalFilterProc` callback each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

reserved1

Must contain 0.

reserved2

Must contain 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

NAPPlayNote

flags Constants

kPickDontMix

Show either all drum kits or all instruments depending on the current instrument. For example, if it's a drum kit, show only drum kits.

kPickSameSynth

Show only instruments from the current synthesizer.

kPickUserInsts

Show modifiable instruments in addition to ROM instruments.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Interface Tools” (V-2918)

Related Java Methods

`quicktime.std.music.ToneDescription.pickInstrument()`

See Also

For a similar function that includes the `kPickEditAllowPick` flag, see `NAPickEditInstrument` (II-1033).

NAPPlayNote

Plays a note with a specified pitch and velocity on the specified note channel.

```
ComponentResult NAPPlayNote (
    NoteAllocator    na,
    NoteChannel      noteChannel,
    long             pitch,
    long             velocity );
```

`na`

You obtain the note allocator identifier from `OpenComponent` (II-1130).

`noteChannel`

The note channel to play the note. You obtain the note channel identifier from the `NANewNoteChannel` (II-1030) or the `NANewNoteChannelFromAtomicInstrument` (II-1031) function.

`pitch`

The pitch at which to play the note. You can specify values as integer pitch values (0–127 where 60 is middle C) or fractional pitch values (256 (0x1.00)

through 32767 (0x7F.FF)). If the pitch is a number from 0 to 127, then it is the MIDI pitch, where 60 is middle C. If the pitch is a positive number above 65535, then the value is a fixed-point pitch value. Thus, microtonal values can be specified. Negative values are not defined and should not be used.

velocity

The velocity with which the key is struck. Typically, this translates directly to volume, but on many synthesizers this also subtly alters the timbre of the tone. A value of 0 is silence; a value of 127 is maximum force.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

```
quicktime.std.music.NoteChannel.playNoteRaw(),
quicktime.std.music.NoteChannel.playNoteCents(),
quicktime.std.music.NoteChannel.playNote()
```

NAPrerollNoteChannel

Attempts to reallocate the note channel if it was invalid previously.

```
ComponentResult NAPrerollNoteChannel (
    NoteAllocator    na,
    NoteChannel      noteChannel );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

noteChannel

Note channel to be re-allocated. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The `NAPrerollNoteChannel` function attempts to reallocate the note channel, if it was invalid previously. It could have been invalid if there were no available voices on any registered music components when the note channel was created.

NARegisterMusicDevice

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.preroll()`

NARegisterMusicDevice

Registers a music component with the note allocator.

```
ComponentResult NARegisterMusicDevice (
    NoteAllocator          na,
    OSType                 synthType,
    Str31                  name,
    SynthesizerConnections *connections );
```

na

You obtain the note allocator identifier from `OpenComponent` (II–1130).

synthType

Subtype of the music component.

name

The synthesizer name. This parameter provides a means of distinguishing multiple instances of the same type of device and is a string that can be displayed to the user. If no value is passed in the name parameter, the name defaults to the name of the music component type. The name appears in the instrument picker dialog box.

connections

A `SynthesizerConnections` (IV–2464) structure that describes the hardware connections to a MIDI device.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Configuration and Utilities” (V–2919)

Related Java Methods

```
quicktime.std.music.NoteAllocator.registerMusicDevice()
```

NAResetNoteChannel

Turns off all currently “on” notes on the note channel and resets all controllers to their default values.

```
ComponentResult NAResetNoteChannel (
    NoteAllocator    na,
    NoteChannel      noteChannel );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

noteChannel

The note channel to reset. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function resets the specified note channel by turning “off” any note currently playing. All controllers are reset to their default state. The effects of the `NAResetNoteChannel` call are propagated down to the allocated part within the appropriate music component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

```
quicktime.std.music.NoteChannel.reset()
```

NASaveMusicConfiguration

Saves the current list of registered devices to a file.

```
ComponentResult NASaveMusicConfiguration (
    NoteAllocator    na );
```

NASendMIDI

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The `NASaveMusicConfiguration` function saves the current list of registered devices to a file. This file is read whenever a note allocator connection is opened, restoring the previously configured list of devices. The list is saved in the QuickTime Preferences file.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Configuration and Utilities” (V–2919)

Related Java Methods

`quicktime.std.music.NoteAllocator.saveMusicConfiguration()`

NASendMIDI

Sends a MIDI music packet to a synthesizer that contains a specific note channel.

```
ComponentResult NASendMIDI (
    NoteAllocator      na,
    NoteChannel        noteChannel,
    MusicMIDIPacket    *mp );
```

na

You obtain the note allocator identifier from the Component Manager’s `OpenComponent` (II–1130) function.

noteChannel

The function sends the packet to the synthesizer that contains this note channel. You obtain the note channel identifier from the `NANewNoteChannel` (II–1030) or the `NANewNoteChannelFromAtomicInstrument` (II–1031) function.

mp

The music packet to be sent.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function sends the MIDI music packet pointed to by the `mp` parameter to the synthesizer that contains the note channel identified by the `noteChannel` parameter. The `na` parameter specifies the note allocator instance to use.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.sendMIDI()`

NSetAtomicInstrument

Initializes a synthesizer part with an atomic instrument.

```
ComponentResult NSetAtomicInstrument (
    NoteAllocator      na,
    NoteChannel        noteChannel,
    AtomicInstrumentPtr instrument,
    long               flags );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

noteChannel

The note channel to apply the atomic instrument to. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

instrument

A pointer to the atomic instrument. This can be a locked, dereferenced atomic instrument.

flags

Flags (see below) that detail how to initialize the part.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`kSetAtomicInstKeepOriginalInstrument`
Keep original sample after expansion.

NASetController

`kSetAtomicInstShareAcrossParts`

Remove the instrument when the application quits.

`kSetAtomicInstCallerTosses`

The caller isn't keeping a copy of the atomic instrument for later calls to `NASetAtomicInstrument`.

`kSetAtomicInstDontPreprocess`

Don't expand the sample. You would only set this bit if you know the instrument is digitally clean or you got it from `MusicGetPartAtomicInstrument` (II-1000).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: "Allocating and Using Note Channels" (V-2916)

Related Java Methods

`quicktime.std.music.NoteChannel.setAtomicInstrument()`

NASetController

Changes the controller setting on a note channel to a specified value.

```
ComponentResult NASetController (
    NoteAllocator    na,
    NoteChannel      noteChannel,
    long             controllerNumber,
    long             controllerValue );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II-1130).

noteChannel

Note channel on which to change controller. You obtain the note channel identifier from `NANewNoteChannel` (II-1030) or `NANewNoteChannelFromAtomicInstrument` (II-1031).

controllerNumber

The controller to set; see "Music Controllers" (IV-2682).

controllerValue

Value for controller setting; typically 0 (0x00.00) to 32767 (0x7F.FF).

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.setControllerRaw()`,
`quicktime.std.music.NoteChannel.setController()`

See Also

For the corresponding get function, see `NAGetController` (II–1022).

NASetDefaultMIDIInput

Initializes an external MIDI device used to receive an external note input.

```
ComponentResult NASETDefaultMIDIInput (
    NoteAllocator      na,
    SynthesizerConnections *sc );
```

na

You obtain the note allocator identifier from the Component Manager’s `OpenComponent` (II–1130) function.

sc

A `SynthesizerConnections` (IV–2464) structure that describes how a MIDI device is connected. The `SynthesizerConnections` fields `clientID`, `inputPortID`, and `outputPortID` are MIDI Manager identifiers. The `midichannel` field is the MIDI system channel value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Configuration and Utilities” (V–2919)

Related Java Methods

`quicktime.std.music.NoteAllocator.setDefaultMIDIInput()`

See Also

For the corresponding get function, see `NAGetDefaultMIDIInput` (II–1023).

NASetInstrumentNumber

NASetInstrumentNumber

Initializes initializes a synthesizer part with the specified instrument.

```
ComponentResult NASetInstrumentNumber (  
    NoteAllocator    na,  
    NoteChannel      noteChannel,  
    long             instrumentNumber );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

noteChannel

Note channel to initialize with the instrument. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

instrumentNumber

Number of the instrument to initialize the part with. This number is unique to each synthesizer. General MIDI synthesizers all share the range 1–128 and 16365 to `kLastDrumKit`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.setInstrumentNumber()`

NASetInstrumentNumberInterruptSafe

Initializes a synthesizer part with the specified instrument during interrupt time.

```
ComponentResult NASetInstrumentNumberInterruptSafe (  
    NoteAllocator    na,  
    NoteChannel      noteChannel,  
    long             instrumentNumber );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

`noteChannel`

Note channel to initialize with the instrument. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

`instrumentNumber`

Number of the instrument to initialize the part with.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

If the instrument is not already loaded when you call `NASetInstrumentNumberInterruptSafe`, you have to wait for the next call to `NATask` (II–1049) for the instrument to become available.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

NASetKnob

Sets a note channel knob to a particular value.

```
ComponentResult NASetKnob (
    NoteAllocator    na,
    NoteChannel      noteChannel,
    long             knobNumber,
    long             knobValue );
```

`na`

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

`noteChannel`

Note channel on which to set the knob value. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

`knobNumber`

Index or ID of the knob to be set.

`knobValue`

Value to set knob to.

NASetNoteChannelBalance

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

quicktime.std.music.NoteChannel.setKnob()

See Also

For the corresponding get function, see NAGetKnob (II–1024).

NASetNoteChannelBalance

Modifies the pan controller setting for a note channel.

```
ComponentResult NASetNoteChannelBalance (
    NoteAllocator    na,
    NoteChannel      noteChannel,
    long             balance );
```

na

You obtain the note allocator identifier by calling OpenComponent (II–1130).

noteChannel

The note channel to be balanced. You obtain the note channel identifier from NANewNoteChannel (II–1030) or NANewNoteChannelFromAtomicInstrument (II–1031).

balance

Specifies how to modify the pan controller setting. Valid values are from –128 to 128 for left to right balance.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

quicktime.std.music.NoteChannel.setBalance()

NASetNoteChannelSoundLocalization

Passes sound localization data to a note channel.

```
ComponentResult NASetNoteChannelSoundLocalization (
    NoteAllocator    na,
    NoteChannel      noteChannel,
    Handle           data );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

noteChannel

The note channel to pass the data to. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

data

Sound localization data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.setSoundLocalization()`

NASetNoteChannelVolume

Sets the volume on the specified note channel.

```
ComponentResult NASetNoteChannelVolume (
    NoteAllocator    na,
    NoteChannel      noteChannel,
    Fixed            volume );
```

na

You obtain the note allocator identifier from the Component Manager’s `OpenComponent` (II–1130) function.

NASuffToneDescription

`noteChannel`

The note channel to reset. You obtain the note channel identifier from the `NANewNoteChannel` (II–1030) or the `NANewNoteChannelFromAtomicInstrument` (II–1031) function.

`volume`

A fixed 16.16 number. `NASetNoteChannelVolume` sets the volume for the note channel, which is different from a `kControllerVolume` setting. Both volume settings allow fractional values of 0.0 to 1.0. Each value modifies the other. For example, a `kControllerVolume` value of 0.5 and a `NASetNoteChannelVolume` value of 0.5 result in a 0.25 volume level.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.setVolume()`

NASuffToneDescription

Initializes a tone description structure with the details of a General MIDI note channel.

```
ComponentResult NASuffToneDescription (  
    NoteAllocator      na,  
    long               gmNumber,  
    ToneDescription    *td );
```

`na`

You obtain the note allocator identifier from the Component Manager’s `OpenComponent` (II–1130) function.

`gmNumber`

A General MIDI instrument number.

`td`

On return, an initialized tone description. The instrument name field will be filled in with the string name for the instrument.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Interface Tools” (V–2918)

Related Java Methods

```
quicktime.std.music.ToneDescription.ToneDescription(),
quicktime.std.music.ToneDescription.stuff(),
quicktime.std.music.NoteChannel.NoteChannel()
```

NATask

Called periodically to allow the note allocator to perform tasks in foreground task time.

```
ComponentResult NATask (
    NoteAllocator    na );
```

`na`

You obtain the note allocator identifier from the Component Manager’s `OpenComponent` (II–1130) function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The `NATask` function calls each registered music component’s `MusicTask` (II–1018) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Configuration and Utilities” (V–2919)

Related Java Methods

```
quicktime.std.music.NoteAllocator.task()
```

NativeEventToMacEvent

Converts Win32 messages to Macintosh events.

NativePathNameToFSSpec

```
long NativeEventToMacEvent (
    void          *nativeEvent,
    EventRecord    *macEvent );
```

nativeEvent

A pointer to a Windows 32 message.

macEvent

A pointer to Macintosh EventRecord (IV–2246) structure to be filled in.

function result Returns noErr if the translation succeeded.

Discussion

This function translates Win32 message types into Macintosh event types and fills in the various other EventRecord (IV–2246) fields based on the source Win32 MSG. Typically, when your application hosts a movie controller, it should call this function to translate a Win32 MSG to an EventRecord, and then pass the resulting EventRecord to MCIsPlayerEvent (II–819) for processing. You should never call this function and then exit early from your WndProc without calling the Windows function DefWindowProc or returning an appropriate result code from your WndProc.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

NativePathNameToFSSpec

Returns an FSSpec (IV–2262) structure for a file from a Windows native pathname.

```
OSErr NativePathNameToFSSpec (
    char          *inName,
    FSSpec         *outFile,
    long           flags );
```

inName

A pointer to the Windows native C string pathname.

outFile

A pointer to FSSpec (IV–2262) structure.

flags

Contains flags that control the conversion. There are no flags currently defined, so you should pass 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. This function returns `noErr` even if the file does not currently exist; in this case, the resulting `FSSpec` (IV–2262) structure is valid for creating the file.

Version Notes

Introduced in QuickTime 3 or earlier. Note that earlier documentation incorrectly stated that if the file does not currently exist, the error `fnfErr` is returned. This is only true for QuickTime 4.0. QuickTime 4.01 and later (as well as all versions of QuickTime 3) return `noErr` even if the file does not currently exist.

Programming Info

C interface file: `QTML.h`

NAUnregisterMusicDevice

Removes a previously registered music component from the note allocator.

```
ComponentResult NAUnregisterMusicDevice (
    NoteAllocator    na,
    long             index );
```

`na`

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

`index`

Synthesizer to unregister. The value is 1 through the registered music component count returned by `NAGetRegisteredMusicDevice` (II–1028).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Note Allocator Configuration and Utilities” (V–2919)

Related Java Methods

`quicktime.std.music.NoteAllocator.unregisterMusicDevice()`

NAUnrollNoteChannel

Marks a note channel as available to be stolen.

NAUseDefaultMIDIInput

```
ComponentResult NARollNoteChannel (
    NoteAllocator    na,
    NoteChannel      noteChannel );
```

na

You obtain the note allocator identifier by calling `OpenComponent` (II–1130).

noteChannel

Note channel to be unrolled. You obtain the note channel identifier from `NANewNoteChannel` (II–1030) or `NANewNoteChannelFromAtomicInstrument` (II–1031).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Allocating and Using Note Channels” (V–2916)

Related Java Methods

`quicktime.std.music.NoteChannel.unroll()`

NAUseDefaultMIDIInput

Defines an entry point to service external MIDI device events.

```
ComponentResult NARollMIDIInput (
    NoteAllocator    na,
    MusicMIDIReadHookUPP readHook,
    long             refCon,
    unsigned long     flags );
```

na

You obtain the note allocator identifier from `OpenComponent` (II–1130).

readHook

A pointer to a `MusicMIDIReadHookProc` (III–2109) callback.

refCon

A reference constant value. The Movie Toolbox passes this reference constant to your `MusicMIDIReadHookProc` callback each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

flags

Must contain 0.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

NAUseDefaultMIDIInput specifies an application’s procedure to service external MIDI events. The specified application’s procedure call, defined by readHook, is called when the external default MIDI device has incoming MIDI data for the application.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Allocating and Using Note Channels” (V–2916)

NewActionsUPP

Allocates a **Universal Procedure Pointer** for ActionsProc (III–2075).

```
ActionsUPP NewActionsUPP (
    ActionsProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewActionsProc.

Programming Info

C interface file: Movies.h

NewCallback

Creates a new **callback event**.

```
QTCallback NewCallback (
    TimeBase    tb,
    short       cbType );
```

NewCallback

tb

The **callback event**'s time base. You obtain this identifier from `NewTimeBase` (II-1119).

cbType

Constants (see below) that specify when the **callback event** is to be invoked. The value of this field governs how the Movie Toolbox interprets the data supplied in the `param1`, `param2`, and `param3` parameters to the `CallMeWhen` (I-67) function. In addition, if the high-order bit of the `cbType` parameter is set to 1 (this bit is defined by the `callBackAtInterrupt` flag), the event can be invoked at interrupt time.

function result

cbType Constants

`callBackAtTime`

Indicates that the event is to be invoked at a specified time.

`callBackAtRate`

Indicates that the event is to be invoked when the rate for the time base reaches a specified value.

`callBackAtTimeJump`

Indicates that the event is to be invoked when the time base's time value changes by an amount that differs from its rate.

`callBackAtExtremes`

Indicates that the event is to be invoked when the time base reaches its start time or its stop time. If the start or stop time of the time base changes, the call back is automatically rescheduled. This is very useful for looping or determining when a movie is complete. You determine when the callback is to be fired with the `triggerAtStart` and `triggerAtStop` constants (see below). Both flags may be set.

`callBackAtInterrupt`

Indicates that the event can be invoked at interrupt time.

callBackAtExtremes Constants

`triggerAtStart`

Call callback at time base start time.

`triggerAtStop`

Call callback at time base stop time.

Special Considerations

The **callback event** created is not active until you schedule it by calling the `CallMeWhen` (I-67) function. You must not call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Time Base Callback Functions” (V–2763)

Related Java Methods

```
quicktime.std.clocks.RateCallback.RateCallback(),
quicktime.std.clocks.TimeCallback.TimeCallback(),
quicktime.std.clocks.TimeJumpCallback.TimeJumpCallback(),
quicktime.std.clocks.ExtremesCallback.ExtremesCallback()
```

NewCharDataHandlerUPP

Undocumented

```
CharDataHandlerUPP NewCharDataHandlerUPP (
    CharDataHandler    userRoutine );
```

`userRoutine`

Undocumented

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

NewCommentHandlerUPP

Undocumented

```
CommentHandlerUPP NewCommentHandlerUPP (
    CommentHandler    userRoutine );
```

`userRoutine`

Undocumented

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).



NewComponentFunctionUPP

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

NewComponentFunctionUPP

Helps native component writers easily write Carbon-compliant components.

```
ComponentFunctionUPP NewComponentFunctionUPP (  
    ProcPtr      userRoutine,  
    ProcInfoType procInfo );
```

userRoutine

A pointer to generic callback function; see Proc (III–2112).

procInfo

The routine’s procedure information.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Components.h

NewComponentMPWorkFunctionUPP

Allocates a **Universal Procedure Pointer** for the ComponentMPWorkFunctionProc (III–2077) callback.

```
ComponentMPWorkFunctionUPP NewComponentMPWorkFunctionUPP (  
    ComponentMPWorkFunctionProcPtr userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewComponentMPWorkFunctionProc.

Programming Info

C interface file: Components.h

NewComponentRoutineUPP

Allocates a **Universal Procedure Pointer** for the ComponentRoutineProc (III–2077) callback.

```
ComponentRoutineUPP NewComponentRoutineUPP (  
    ComponentRoutineProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewComponentRoutineProc.

Programming Info

C interface file: Components.h

NewDataHCompletionUPP

Allocates a **Universal Procedure Pointer** for the DataHCompletionProc (III–2078) callback.

```
DataHCompletionUPP NewDataHCompletionUPP (  
    DataHCompletionProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

NewDoMCActionUPP

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewDataHCompletionProc.

Programming Info

C interface file: QuickTimeComponents.h

NewDoMCActionUPP

Allocates a **Universal Procedure Pointer** for the DoMCActionProc (III–2082) callback.

```
DoMCActionUPP NewDoMCActionUPP (  
    DoMCActionProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewDoMCActionProc.

Programming Info

C interface file: Movies.h

NewEndDocumentHandlerUPP

Undocumented

```
EndDocumentHandlerUPP NewEndDocumentHandlerUPP (  
    EndDocumentHandler    userRoutine );
```

userRoutine

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

NewEndElementHandlerUPP

Undocumented

```
EndElementHandlerUPP NewEndElementHandlerUPP (  
    EndElementHandler    userRoutine );
```

userRoutine

Undocumented

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

NewFilePlayCompletionUPP

Obsolete.

```
FilePlayCompletionUPP NewFilePlayCompletionUPP (  
    FilePlayCompletionProcPtr    userRoutine );
```

Version Notes

Introduced in QuickTime 4.1 to replace NewFilePlayCompletionProc. Macintosh

Note: Not supported by **CarbonLib**.

Programming Info

C interface file: Sound.h

NewGetMissingComponentResourceUPP

Allocates a **Universal Procedure Pointer** for the GetMissingComponentResourceProc (III–2084) callback.

NewGetMovieUPP

```
GetMissingComponentResourceUPP NewGetMissingComponentResourceUPP (  
    GetMissingComponentResourceProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewGetMissingComponentResourceProc.

Programming Info

C interface file: Components.h

NewGetMovieUPP

Allocates a **Universal Procedure Pointer** for the GetMovieProc (III–2085) callback.

```
GetMovieUPP NewGetMovieUPP (  
    GetMovieProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewGetMovieProc.

Programming Info

C interface file: Movies.h

NewICMAlignmentUPP

Allocates a **Universal Procedure Pointer** for the ICMAlignmentProc (III–2086) callback.

```
ICMAlignmentUPP NewICMAlignmentUPP (
    ICMAlignmentProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewICMAlignmentProc.

Programming Info

C interface file: ImageCompression.h

NewICMCompletionUPP

Allocates a **Universal Procedure Pointer** for the ICMCompletionProc (III–2087) callback.

```
ICMCompletionUPP NewICMCompletionUPP (
    ICMCompletionProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewICMCompletionProc.

Programming Info

C interface file: ImageCompression.h

NewICMConvertDataFormatUPP

Allocates a **Universal Procedure Pointer** for the ICMConvertDataFormatProc (III–2088) callback.

NewICMCursorShieldedUPP

```
ICMConvertDataFormatUPP NewICMConvertDataFormatUPP (  
    ICMConvertDataFormatProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewICMConvertDataFormatProc.

Programming Info

C interface file: ImageCompression.h

NewICMCursorShieldedUPP

Allocates a **Universal Procedure Pointer** for the ICMCursorShieldedProc (III–2089) callback.

```
ICMCursorShieldedUPP NewICMCursorShieldedUPP (  
    ICMCursorShieldedProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewICMCursorShieldedProc.

Programming Info

C interface file: ImageCompression.h

NewICMDataUPP

Allocates a **Universal Procedure Pointer** for the ICMDataProc (III–2090) callback.

```
ICMDataUPP NewICMDataUPP (
    ICMDataProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewICMDataProc.

Programming Info

C interface file: ImageCompression.h

NewICMFlushUPP

Allocates a **Universal Procedure Pointer** for the ICMFlushProc (III–2091) callback.

```
ICMFlushUPP NewICMFlushUPP (
    ICMFlushProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewICMFlushProc.

Programming Info

C interface file: ImageCompression.h

NewICMMemoryDisposedUPP

Allocates a **Universal Procedure Pointer** for the ICMMemoryDisposedProc (III–2092) callback.

NewICMProgressUPP

```
ICMemoryDisposedUPP NewICMemoryDisposedUPP (  
    ICMemoryDisposedProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewICMemoryDisposedProc.

Programming Info

C interface file: ImageCompression.h

NewICMProgressUPP

Allocates a **Universal Procedure Pointer** for the ICMProgressProc (III–2093) callback.

```
ICMProgressUPP NewICMProgressUPP (  
    ICMProgressProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewICMProgressProc.

Programming Info

C interface file: ImageCompression.h

NewImageCodecDrawBandCompleteUPP

Allocates a **Universal Procedure Pointer** for an ImageCodecDrawBandCompleteProc (III–2094) callback.

```
ImageCodecDrawBandCompleteUPP NewImageCodecDrawBandCompleteUPP (  
    ImageCodecDrawBandCompleteProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: ImageCodec.h

NewImageCodecMPDrawBandUPP

Allocates a **Universal Procedure Pointer** for the ImageCodecMPDrawBandProc (III–2095) callback.

```
ImageCodecMPDrawBandUPP NewImageCodecMPDrawBandUPP (  
    ImageCodecMPDrawBandProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewImageCodecMPDrawBandProc.

Programming Info

C interface file: ImageCodec.h

NewImageCodecTimeTriggerUPP

Allocates a **Universal Procedure Pointer** for the ImageCodecTimeTriggerProc (III–2096) callback.

```
ImageCodecTimeTriggerUPP NewImageCodecTimeTriggerUPP (  
    ImageCodecTimeTriggerProcPtr    userRoutine );
```

NewImageGWorld

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewImageCodecTimeTriggerProc`.

Programming Info

C interface file: `ImageCodec.h`

NewImageGWorld

Creates an offscreen graphics world.

```
OSErr NewImageGWorld (
    GWorldPtr          *gworld,
    ImageDescriptionHandle idh,
    GWorldFlags         flags );
```

`gworld`

A pointer to a graphic world created using the width, height, depth, and color table specified in the `ImageDescription` (IV–2285) structure pointed to in the `idh` parameter.

`idh`

A handle to an `ImageDescription` (IV–2285) structure that contains information for the graphics world pointed to by the `gworld` parameter.

`flags`

Graphics world creation flags (see below). The `pixPurge`, `noNewDevice`, `useTempMem`, `keepLocal`, `pixelsPurgeable`, and `pixelsLocked` flags are commands to this function; the others are returned by this function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`pixPurge`

Make the base address for the offscreen pixel image purgeable.

noNewDevice

Do not create a new offscreen GDevice (IV-2264) structure; instead, use either the GDevice record you specify or the GDevice record for a video card on the user's system.

useTempMem

Create the base address for an offscreen pixel image in temporary memory. You generally should not use this flag. You should use temporary memory only for fleeting purposes.

keepLocal

Create a pixel image in main memory where it cannot be cached to a graphics accelerator card.

pixelsPurgeable

Make the base address for an offscreen pixel map purgeable. If you use this function without passing it this flag, it makes the base address for an offscreen pixel map unpurgeable.

pixelsLocked

Lock the base address for an offscreen pixel image. If you use this function without passing it this flag, it unlocks the offscreen pixel image.

mapPix

Returned if this function remapped the colors in the offscreen pixel map to a new color table.

newDepth

Returned if this function translated the offscreen pixel map to a different pixel depth.

alignPix

Returned if this function realigned the offscreen pixel image to an onscreen window.

newRowBytes

Returned if this function changed the rowBytes field of the PixMap record for the offscreen graphics world.

reallocPix

Returned if this function reallocated the base address for the offscreen pixel image. Your application should then reconstruct the pixel image or draw directly in a window instead of preparing the image in an offscreen graphics world.

clipPix

Returned if this function clipped the pixel image.

NewMCActionFilterUPP

stretchPix

Returned if this function stretched or shrank the offscreen image.

ditherPix

Returned if this function **dithered** the offscreen image.

gwFlagErr

Returned if this function was unsuccessful and the offscreen graphics world was left unchanged.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Working With Graphics Devices and Graphics Worlds” (V–2798)

Related Java Methods

quicktime.qd.QDGraphics.QDGraphics(),

quicktime.std.image.ImageDescription.newGWorld()

NewMCActionFilterUPP

Allocates a **Universal Procedure Pointer** for the MCActionFilterProc (III–2096) callback.

```
MCActionFilterUPP NewMCActionFilterUPP (  
    MCActionFilterProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewMCActionFilterProc.

Programming Info

C interface file: Movies.h

NewMCActionFilterWithRefConUPP

Allocates a **Universal Procedure Pointer** for the `MCActionFilterWithRefConProc` (III–2097) callback.

```
MCActionFilterWithRefConUPP NewMCActionFilterWithRefConUPP (
    MCActionFilterWithRefConProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewMCActionFilterWithRefConProc`.

Programming Info

C interface file: `Movies.h`

NewMovie

Creates a new movie in memory.

```
Movie NewMovie (
    long    flags );
```

`flags`

Flags (see below) that specify control information for the new movie. Be sure to set unused flags to 0.

function result The identifier for the new movie. If `NewMovie` fails, the returned identifier is set to `NIL`. You can use `GetMoviesError` (I–492) to obtain the error result, or `noErr` if there was no error. See “Error Codes” (IV–2663).

Flags Constants

`newMovieActive`

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. A movie that does not have any tracks can still be active. When the Movie Toolbox tries to play the movie, no images are displayed, because

NewMovie

there is no movie data. You can make a movie active or inactive by calling `SetMovieActive` (III–1609).

`newMovieDontAutoAlternate`

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie; you must enable tracks yourself.

Discussion

You can use `NewMovie` to create a new empty movie, which contains no tracks. The Movie Toolbox initializes the data structures for the new movie. Your application assigns the data to the movie by calling the functions that are described in `NewMovieTrack` (II–1092).

The Movie Toolbox sets many movie characteristics to default values. If you want to change these defaults, your application must call other Movie Toolbox functions. For example, the Movie Toolbox sets the movie's graphics world to the one that is active when you call `NewMovie`. To change the graphics world for the new movie, your application should use `SetMovieGWorld` (III–1619). The default QuickTime movie time scale is 600 units per second; however, this number may change in the future. The default time scale was chosen because it is convenient for working with common video frame rates of 30, 25, 24, 15, 12, 10, and 8.

Special Considerations

The Movie Toolbox automatically sets the movie's graphics world based upon the current graphics port. Be sure that your application's graphics port is valid before you call this function. Use this function only if you have not created a new movie and movie file by calling `CreateMovieFile` (I–145).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Functions” (V–2713)

Related Java Methods

`quicktime.std.movies.Movie.Movie()`

See Also

If your application is loading a movie from an existing file, use either `NewMovieFromFile` (II–1080) or `NewMovieFromDataFork` (II–1075). `NewMovieFromFile` works with the file reference number you obtain from `OpenMovieFile` (II–1133); `NewMovieFromDataFork` works with movies stored in a Macintosh document file's data fork.

NewMovieController

Locates a movie controller component and assigns a movie to that controller.

```
ComponentInstance NewMovieController (
    Movie          theMovie,
    const Rect     *movieRect,
    long           someFlags );
```

theMovie

The movie to be associated with the movie controller.

movieRect

A pointer to the Rect (IV–2399) structure that is to define the display boundaries of the movie and its controller.

someFlags

Contains flags (see below) that control the operation. If you set these flags to 0, the movie controller component centers the movie in the rectangle specified by the *movieRect* parameter and scales the movie to fit in that rectangle. The control portion of the controller is also placed within that rectangle. You may control how the movie and the control are drawn by setting one or more flags to 1.

function result The ID of the new controller.

someFlags Constants

mcTopLeftMovie

If this flag is set to 1, the movie controller component places the movie into the upper-left corner of the display rectangle specified by the *movieRect* parameter. The component scales the movie to fit into the rectangle. Note that the control portion of the controller may fall outside of the rectangle, depending upon the results of the scaling operation.

mcScaleMovieToFit

If this flag is set to 1, the movie controller component resizes the movie to fit into the display rectangle specified by the *movieRect* parameter after it places the control portion of the controller into the rectangle. If you set this flag and the *mcScaleMovieToFit* flag to 1, the movie controller component resizes the movie to fit into the specified rectangle and places the control portion of the controller outside of the rectangle.

mcWithBadge

Controls whether the movie controller uses a badge. If you set this flag to 1, the movie controller component displays the movie with a badge whenever the

NewMovieDrawingCompleteUPP

controller portion is not displayed. If you set this flag to 0, the movie controller component does not use a badge.

`mcNotVisible`

Controls whether the controller portion is visible. If you set this flag to 0, the movie controller component displays the controller along with the movie. If you set this flag to 1, the component does not display the controller. If you have set the `mcWithBadge` flag to 1, specifying that the component uses a badge, the component displays a badge whenever the controller is not visible.

`mcWithFrame`

Specifies whether the component displays a frame around the movie as part of the controller. If you set this flag to 1, the component displays a frame around the movie, including the movie's name. If you set this flag to 0, the component does not display a frame as part of the controller.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Associating Movies With Controllers” (V–2774)

Related Java Methods

`quicktime.std.movies.MovieController.MovieController()`,

`quicktime.std.movies.MultiMovieController.MultiMovieController()`

NewMovieDrawingCompleteUPP

Allocates a **Universal Procedure Pointer** for the `MovieDrawingCompleteProc` (III–2100) callback.

```
MovieDrawingCompleteUPP NewMovieDrawingCompleteUPP (  
    MovieDrawingCompleteProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewMovieDrawingCompleteProc`.

Programming Info

C interface file: `Movies.h`

NewMovieEditState

Creates an edit state.

```
MovieEditState NewMovieEditState (
    Movie    theMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result A pointer to a `MovieEditStateRecord` (IV–2313) structure. The edit state contains all the information describing a movie’s content, including the current selection, the movie’s tracks, and the media data associated with those tracks.

Special Considerations

You must dispose of a movie’s `MovieEditStateRecord` structures, using `DisposeMovieEditState` (I–273), before you dispose of the movie itself.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Undo for Movies” (V–2748)

Related Java Methods

`quicktime.std.movies.Movie.newEditState()`

NewMovieExecuteWiredActionsUPP

Allocates a **Universal Procedure Pointer** for the `MovieExecuteWiredActionsProc` (III–2101) callback.

```
MovieExecuteWiredActionsUPP NewMovieExecuteWiredActionsUPP (
    MovieExecuteWiredActionsProcPtr    userRoutine );
```

NewMovieExportGetDataUPP

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewMovieExecuteWiredActionsProc`.

Programming Info

C interface file: `Movies.h`

NewMovieExportGetDataUPP

Allocates a **Universal Procedure Pointer** for the `MovieExportGetDataProc` (III–2101) callback.

```
MovieExportGetDataUPP NewMovieExportGetDataUPP (  
    MovieExportGetDataProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewMovieExportGetDataProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewMovieExportGetPropertyUPP

Allocates a **Universal Procedure Pointer** for the `MovieExportGetPropertyProc` (III–2102) callback.

```
MovieExportGetPropertyUPP NewMovieExportGetPropertyUPP (  
    MovieExportGetPropertyProcPtr    userRoutine );
```


`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewMovieExportGetPropertyProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewMovieFromDataFork

Retrieves a movie that is stored anywhere in the data fork of a specified Macintosh file.

```
OSErr NewMovieFromDataFork (
    Movie      *theMovie,
    short      fRefNum,
    long       fileOffset,
    short      newMovieFlags,
    Boolean     *dataRefWasChanged );
```

`theMovie`

A pointer to a field that is to receive the new movie’s identifier. If the function cannot load the movie, the returned identifier is set to `NIL`.

`fRefNum`

A file reference number to a file that is already open.

`fileOffset`

The starting file offset of the atom in the data fork of the file specified by the `fRefNum` parameter.

`newMovieFlags`

Flags (see below) that control characteristics of the new movie.

`dataRefWasChanged`

A pointer to a `Boolean` value. The Movie Toolbox sets the value to `TRUE` if any of the movie’s data references were changed. Use `UpdateMovieResource` (III–1983) to preserve these changes. If you do not want to receive this information, set the `dataRefWasChanged` parameter to `NIL`.

NewMovieFromDataFork

function result If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access error returns such as this through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

newMovieFlags Constants

`newMovieActive`

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. A movie that does not have any tracks can still be active. When the Movie Toolbox tries to play the movie, no images are displayed, because there is no movie data. You can make a movie active or inactive by calling the `SetMovieActive` (III–1609) function.

`newMovieDontAutoAlternate`

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie; you must enable tracks yourself.

`newMovieDontResolveDataRefs`

Controls how completely the Movie Toolbox resolves data references in the movie **resource**. If you set this flag to 0, the Movie Toolbox tries to completely resolve all data references in the resource. This may involve searching for files on remote volumes. If you set this flag to 1, the Movie Toolbox only looks in the specified file. If the Movie Toolbox cannot completely resolve all the data references, it still returns a valid movie identifier. In this case, the Movie Toolbox also sets the current error value to `couldNotResolveDataRef`.

`newMovieDontAskUnresolvedDataRefs`

Controls whether the Movie Toolbox asks the user to locate files. If you set this flag to 0, the Movie Toolbox asks the user to locate files that it cannot find on available volumes. If the Movie Toolbox cannot locate a file even with the user’s help, the function returns a valid movie identifier and sets the current error value to `couldNotResolveDataRef`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Saving Movies” (V–2715)

Related Java Methods

`quicktime.std.movies.Movie.fromDataFork()`

NewMovieFromDataFork64

Provides a 64-bit version of NewMovieFromDataFork (II–1075).

```
OSErr NewMovieFromDataFork64 (
    Movie      *theMovie,
    long       fRefNum,
    const wide *fileOffset,
    short      newMovieFlags,
    Boolean     *dataRefWasChanged );
```

theMovie

A pointer to a field that is to receive the new movie’s identifier. If the function cannot load the movie, the returned identifier is set to NIL.

fRefNum

A file reference number to a file that is already open.

fileOffset

A pointer to the starting file offset of the atom in the data fork of the file specified by the *fRefNum* parameter.

newMovieFlags

Flags (see below) that control characteristics of the new movie.

dataRefWasChanged

A pointer to a Boolean value. The Movie Toolbox sets the value to TRUE if any of the movie’s data references were changed. Use `UpdateMovieResource` (III–1983) to preserve these changes. If you do not want to receive this information, set the *dataRefWasChanged* parameter to NIL.

function result

If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access error returns such as this through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

newMovieFlags Constants

newMovieActive

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. A movie that does not have any tracks can still be active. When the Movie Toolbox tries to play the movie, no images are displayed, because there is no movie data. You can make a movie active or inactive by calling the `SetMovieActive` (III–1609) function.

NewMovieFromDataRef

`newMovieDontAutoAlternate`

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie; you must enable tracks yourself.

`newMovieDontResolveDataRefs`

Controls how completely the Movie Toolbox resolves data references in the movie **resource**. If you set this flag to 0, the Movie Toolbox tries to completely resolve all data references in the resource. This may involve searching for files on remote volumes. If you set this flag to 1, the Movie Toolbox only looks in the specified file. If the Movie Toolbox cannot completely resolve all the data references, it still returns a valid movie identifier. In this case, the Movie Toolbox also sets the current error value to `couldNotResolveDataRef`.

`newMovieDontAskUnresolvedDataRefs`

Controls whether the Movie Toolbox asks the user to locate files. If you set this flag to 0, the Movie Toolbox asks the user to locate files that it cannot find on available volumes. If the Movie Toolbox cannot locate a file even with the user's help, the function returns a valid movie identifier and sets the current error value to `couldNotResolveDataRef`.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

NewMovieFromDataRef

Creates a movie from any device with a corresponding data handler.

```
OSErr NewMovieFromDataRef (
    Movie      *m,
    short      flags,
    short      *id,
    Handle     dataRef,
    OSType     dataRefType );
```

m

A pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `NIL`.

flags

Flags (see below) that control the operation of this function. Be sure to set unused flags to 0.

id

A pointer to the field that specifies the **resource** containing the movie data that is to be loaded. If the field referred to by the `id` parameter is set to 0, the Movie Toolbox loads the first movie resource it finds in the specified file. The toolbox then returns the movie's resource ID number in the field referred to by the `id` parameter. An enumerated constant (see below) is available.

dataRef

The default data reference. This parameter contains a handle to the information that identifies the file to be used to resolve any data references and as a starting point for any Alias Manager searches. The type of information stored in the handle depends upon the value of the `dataRefType` parameter. For example, if your application is loading the movie from a file, you would refer to the file's **alias** in this parameter and set the `dataRefType` parameter to `rAliasType`. If you do not want to identify a default data reference, set the parameter to `NIL`.

dataRefType

The type of data reference. If the data reference is an **alias**, you must set the parameter to `rAliasType`, indicating that the reference is an alias.

function result If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access error returns such as this through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

flags Constants**newMovieActive**

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. You can make a movie active or inactive by calling the `SetMovieActive` (III-1609) function.

newMovieDontResolveDataRefs

Controls how completely the Movie Toolbox resolves data references in the movie **resource**. If you set this flag to 0, the toolbox tries to completely resolve all data references in the resource. This may involve searching for files on remote volumes. If you set this flag to 1, the toolbox only looks in the specified data reference. If the toolbox cannot completely resolve all the data references, it still returns a valid movie identifier. In this case, the toolbox also sets the current error value to `couldNotResolveDataRef`.

NewMovieFromFile

`newMovieDontAskUnresolvedDataRefs`

Controls whether the Movie Toolbox asks the user to locate files. If you set this flag to 0, the toolbox asks the user to locate files that it cannot find on available volumes. If the toolbox cannot locate a file, even with the user's help, the function returns a valid movie identifier and sets the current error value to `couldNotResolveDataRef`.

`newMovieDontAutoAlternate`

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie; you must enable and disable tracks yourself.

id Constants

`movieInDataForkResID`

The movie was loaded from the data fork. If the **resource** was stored in the file's data fork, the Movie Toolbox sets the returned value to this constant. In this case, you cannot add a movie resource to the file unless you create a resource fork in the movie file.

Discussion

This function is intended for use by specialized applications that need to instantiate movies from devices not visible to the file system. Most applications should continue to use `NewMovieFromFile` (II-1080). You are not restricted to instantiating a movie from a file stored on a Macintosh **HFS** volume. With this function, you can instantiate a movie from any device.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Movie Functions" (V-2713)

Related Java Methods

`quicktime.std.movies.Movie.fromDataRef()`

NewMovieFromFile

Creates a new movie from certain file types that do not contain movie **resources** (AIFF, MPEG, etc).

```
OSErr NewMovieFromFile (
    Movie          *theMovie,
    short          resRefNum,
```

```

short      *resId,
StringPtr  resName,
short      newMovieFlags,
Boolean    *dataRefWasChanged );

```

theMovie

A pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `NIL`.

resRefNum

The movie file from which the movie is to be loaded. Your application obtains this value from the `OpenMovieFile` (II–1133) function.

resId

A pointer to a field that specifies the **resource** containing the movie data that is to be loaded. If the field referred to by the `resId` parameter is set to 0, the Movie Toolbox loads the first movie resource it finds in the specified file. The Movie Toolbox then returns the movie's resource ID number in the field referred to by the `resId` parameter. An enumerated constant (see below) is available.

resName

A pointer to a character string that is to receive the name of the movie **resource** that is loaded. If you set the `resName` parameter to `NIL`, the Movie Toolbox does not return the resource name.

newMovieFlags

Flags (see below) that control the operation of `NewMovieFromFile`. Be sure to set unused flags to 0.

dataRefWasChanged

A pointer to a Boolean value. The Movie Toolbox sets the value to `TRUE` if any references were changed. Use `UpdateMovieResource` (III–1983) to preserve these changes. Set this parameter to `NIL` if you don't want to receive this information. See `NewMovieTrack` (II–1092) for more information about data references.

function result If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access error returns such as this through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

resId Constant

movieInDataForkResID

Forces the movie to come out of the data fork. If the **resource** was stored in the file's data fork, the Movie Toolbox sets the returned value to this constant. In this case, you can only add a movie to the data fork of the file if you create a

NewMovieFromFile

resource fork in the movie file. If the `resId` parameter is set to `NIL`, the Movie Toolbox loads the first movie resource it finds in the specified file and does not return that resource's ID number.

newMovieFlags Constants

`newMovieActive`

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. You can make a movie active or inactive by calling `SetMovieActive` (III–1609).

`newMovieDontResolveDataRefs`

Controls how completely the Movie Toolbox resolves data references in the movie **resource**. If you set this flag to 0, the Movie Toolbox tries to completely resolve all data references in the resource. This may involve searching for files on multiple volumes. If you set this flag to 1, the Movie Toolbox only looks in the specified file. If the Movie Toolbox cannot completely resolve all the data references, it still returns a valid movie identifier. In this case, the Movie Toolbox also sets the current error value to `couldNotResolveDataRef`.

`newMovieDontAskUnresolvedDataRefs`

Controls whether the Movie Toolbox asks the user to locate files. If you set this flag to 0, the Movie Toolbox asks the user to locate files that it cannot find. If the Movie Toolbox cannot locate a file even with the user's help, the function returns a valid movie identifier and sets the current error value to `couldNotResolveDataRef`.

`newMovieDontAutoAlternate`

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie; you must enable tracks yourself.

Discussion

The Movie Toolbox sets many movie characteristics to default values. If you want to change these defaults, your application must call other Movie Toolbox functions. For example, the Movie Toolbox sets the movie's graphics world to the one that is active when you call `NewMovieFromFile`. To change the graphics world for the new movie, your application should use `SetMovieGWorld` (III–1619).

The following is an example of using this function:

```
// NewMovieFromFile coding example
// See "Discovering QuickTime," page 385
Movie MyGetMovie (void)
{
    OSErr              nErr;
```



```

SFTypelist          types = {MovieFileType, 0, 0, 0};
StandardFileReply   sfr;
Movie               movie = NIL;
short               nFileRefNum;

StandardGetFilePreview(NIL, 1, types, &sfr);
if (sfr.sfGood) {
    nErr = OpenMovieFile(&sfr.sfFile, &nFileRefNum, fsRdPerm);
    if (nErr == noErr) {
        short          nResID = 0;          //We want the first movie.
        Str255          strName;
        Boolean          bWasChanged;

        nErr = NewMovieFromFile(&movie, nFileRefNum, &nResID, strName,
                                newMovieActive, &bWasChanged);
        CloseMovieFile(nFileRefNum);
    }
}
return movie;
}

```

Special Considerations

This function works with some files that don't contain movie **resources**. When it encounters a file that does not contain a movie resource, it tries to find a movie import component that can understand the data and create a movie. It also works for MPEG, uLaw (.AU), and Wave (.WAV) file types. In some cases, the data in a file is already sufficiently well formatted for QuickTime or its components to understand. For example, the AIFF movie data import component can understand AIFF sound files and import the sound data into a QuickTime movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Movie Functions" (V-2713)

Related Java Methods

`quicktime.std.movies.Movie.fromFile()`

See Also

Your application can use `NewMovieFromHandle` (II-1084) to load a movie from a handle.

NewMovieFromHandle

Creates a movie in memory from a movie **resource** or a handle you obtained from PutMovieIntoHandle (II–1151).

```
OSErr NewMovieFromHandle (
    Movie      *theMovie,
    Handle     h,
    short      newMovieFlags,
    Boolean    *dataRefWasChanged );
```

theMovie

A pointer to a field that is to receive the new movie’s identifier. If the function cannot load the movie, the returned identifier is set to NIL.

h

A handle to the movie **resource** from which the movie is to be loaded.

newMovieFlags

Flags (see below) that control the operation of NewMovieFromHandle. Be sure to set unused flags to 0.

dataRefWasChanged

A pointer to a Boolean value. The toolbox sets the value to TRUE if any references were changed. Set the dataRefWasChanged parameter to NIL if you don’t want to receive this information.

function result If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to couldNotResolveDataRef. You can access error returns such as this through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

newMovieFlags Constants

newMovieActive

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. You can make a movie active or inactive by calling SetMovieActive (III–1609).

newMovieDontResolveDataRefs

Controls how completely the Movie Toolbox resolves data references in the movie **resource**. If you set this flag to 0, the toolbox tries to completely resolve all data references in the resource. This may involve searching for files on remote volumes. If you set this flag to 1, the Movie Toolbox only looks in the specified file. If the Movie Toolbox cannot completely resolve all the data

references, it still returns a valid movie identifier. In this case, the Movie Toolbox also sets the current error value to `couldNotResolveDataRef`.

`newMovieDontAskUnresolvedDataRefs`

Controls whether the Movie Toolbox asks the user to locate files. If you set this flag to 0, the Movie Toolbox asks the user to locate files that it cannot find on available volumes. If the Movie Toolbox cannot locate a file even with the user's help, the function returns a valid movie identifier and sets the current error value to `couldNotResolveDataRef`.

`newMovieDontAutoAlternate`

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie; you must enable tracks yourself.

Discussion

The Movie Toolbox sets many movie characteristics to default values. If you want to change these defaults, your application must call other Movie Toolbox functions. For example, the Movie Toolbox sets the movie's graphics world to the one that is active when you call `NewMovieFromHandle`. To change the graphics world for the new movie, your application should use `SetMovieGWorld` (III-1619).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Movie Functions" (V-2713)

Related Java Methods

`quicktime.std.movies.Movie.fromHandle()`

See Also

Your application can use `NewMovieFromFile` (II-1080) to load a movie from a movie file that was opened with `OpenMovieFile` (II-1133).

NewMovieFromScrap

Creates a movie from the contents of the scrap.

```
Movie NewMovieFromScrap (
    long    newMovieFlags );
```

NewMovieFromScrap

newMovieFlags

Flags (see below) that control the operation of the NewMovieFromScrap function. Be sure to set unused flags to 0.

function result The identifier for the new movie. If NewMovieFromScrap fails, or if there is no movie in the scrap, the returned identifier is set to NIL. You can use GetMoviesError (I-492) to obtain the error result, or noErr if there was no error. See “Error Codes” (IV-2663).

newMovieFlags Constants

newMovieActive

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. A movie that does not have any tracks can still be active. When the Movie Toolbox tries to play the movie, no images are displayed, because there is no movie data. Unless you set this flag, you should call the SetMovieActive (III-1609) function to play a movie.

newMovieDontResolveDataRefs

Controls how completely the Movie Toolbox resolves data references in the movie **resource**. If you set this flag to 0, the toolbox tries to completely resolve all data references in the resource. This may involve searching for files on remote volumes. If you set this flag to 1, the Movie Toolbox only looks in the specified file. If the Movie Toolbox cannot completely resolve all the data references, it still returns a valid movie identifier. In this case, the Movie Toolbox also sets the current error value to `couldNotResolveDataRef`.

newMovieDontAskUnresolvedDataRefs

Controls whether the Movie Toolbox asks the user to locate files. If you set this flag to 0, the Movie Toolbox asks the user to locate files that it cannot find on available volumes. If the Movie Toolbox cannot locate a file even with the user’s help, the function returns a valid movie identifier and sets the current error value to `couldNotResolveDataRef`.

newMovieDontAutoAlternate

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie; you must enable tracks yourself.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Movie Editing Functions” (V-2746)

Related Java Methods

```
quicktime.std.movies.Movie.fromScrap()
```

NewMovieFromUserProc

Creates a movie from data that you provide.

```
OSErr NewMovieFromUserProc (
    Movie          *m,
    short          flags,
    Boolean        *dataRefWasChanged,
    GetMovieUPP    getProc,
    void           *refCon,
    Handle         defaultDataRef,
    OSType         dataRefType );
```

m

A pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to NIL.

flags

Flags (see below) that control the operation of the NewMovieFromUserProc function. Be sure to set unused flags to 0.

dataRefWasChanged

A pointer to a Boolean value. The Toolbox sets the value to TRUE if any references were changed. Use UpdateMovieResource (III–1983) to preserve these changes. Set the dataRefWasChanged parameter to NIL if you don't want to receive this information.

getProc

A **Universal Procedure Pointer** that accesses a GetMovieProc (III–2085) callback, which is responsible for providing the movie data to the Movie Toolbox.

refCon

A reference constant (defined as a void pointer). This is the same value you provided to the Movie Toolbox when you called NewMovieFromUserProc. Use this parameter to point to a data structure containing any information your callback needs.

defaultDataRef

The default data reference. This parameter contains a handle to the information that identifies the file to be used to resolve any data references and as a starting point for any Alias Manager searches. The type of information stored in the handle depends upon the value of the dataRefType parameter. For example, if

NewMovieFromUserProc

your application is loading the movie from a file, you would refer to the file's **alias** in the `defaultDataRef` parameter, and set the `dataRefType` parameter to `rAliasType`. If you don't want to identify a default data reference, set the parameter to `NIL`.

`dataRefType`

The type of data reference. If the data reference is an **alias**, you must set the parameter to `rAliasType`, indicating that the reference is an alias.

function result If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access error returns such as this through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

flags Constants

`newMovieActive`

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. You can make a movie active or inactive by calling `SetMovieActive` (III-1609).

`newMovieDontResolveDataRefs`

Controls how completely the Movie Toolbox resolves data references in the movie **resource**. If you set this flag to 0, the toolbox tries to completely resolve all data references in the resource. This may involve searching for files on remote volumes. If you set this flag to 1, the toolbox only looks in the specified data reference. If the toolbox cannot completely resolve all the data references, it still returns a valid movie identifier. In this case, the toolbox also sets the current error value to `couldNotResolveDataRef`.

`newMovieDontAskUnresolvedDataRefs`

Controls whether the Movie Toolbox asks the user to locate files. If you set this flag to 0, the toolbox asks the user to locate files that it cannot find on available volumes. If the toolbox cannot locate a file even with the user's help, the function returns a valid movie identifier and sets the current error value to `couldNotResolveDataRef`.

`newMovieDontAutoAlternate`

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the toolbox does not automatically select tracks for the movie; you must enable and disable tracks yourself.

Discussion

Normally, when a movie is loaded from a file (for example, by means of `NewMovieFromFile` (II-1080)), the Movie Toolbox uses that file as the default data reference. Since this function does not require a file specification, your application should specify the file to be used as the default data reference using the `defaultDataRef` and `dataRefType` parameters.

Special Considerations

The Movie Toolbox automatically sets the movie's graphics world based upon the current graphics port. Be sure that your application's graphics world is valid before you call this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Movie Functions" (V-2713)

NewMoviePrePrerollCompleteUPP

Allocates a **Universal Procedure Pointer** for the `MoviePrePrerollCompleteProc` (III-2104) callback.

```
MoviePrePrerollCompleteUPP NewMoviePrePrerollCompleteUPP (
    MoviePrePrerollCompleteProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see "Universal Procedure Pointers" (IV-2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewMoviePrePrerollCompleteProc`.

Programming Info

C interface file: `Movies.h`

NewMoviePreviewCallOutUPP

NewMoviePreviewCallOutUPP

Allocates a **Universal Procedure Pointer** for the MoviePreviewCallOutProc (III–2105) callback.

```
MoviePreviewCallOutUPP NewMoviePreviewCallOutUPP (  
    MoviePreviewCallOutProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewMoviePreviewCallOutProc.

Programming Info

C interface file: Movies.h

NewMovieProgressUPP

Allocates a **Universal Procedure Pointer** for the MovieProgressProc (III–2105) callback.

```
MovieProgressUPP NewMovieProgressUPP (  
    MovieProgressProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewMovieProgressProc.

Programming Info

C interface file: Movies.h

NewMovieRgnCoverUPP

Allocates a **Universal Procedure Pointer** for the MovieRgnCoverProc (III–2108) callback.

```
MovieRgnCoverUPP NewMovieRgnCoverUPP (
    MovieRgnCoverProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewMovieRgnCoverProc.

Programming Info

C interface file: Movies.h

NewMoviesErrorUPP

Allocates a **Universal Procedure Pointer** for the MoviesErrorProc (III–2109) callback.

```
MoviesErrorUPP NewMoviesErrorUPP (
    MoviesErrorProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Version Notes

Introduced in QuickTime 4.1. Replaces NewMoviesErrorProc.

Programming Info

C interface file: Movies.h

NewMovieTrack

NewMovieTrack

Creates a new movie track, without a media.

```
Track NewMovieTrack (  
    Movie    theMovie,  
    Fixed    width,  
    Fixed    height,  
    short    trackVolume );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

width

A fixed number denoting the display width of the track, in pixels.

height

A fixed number denoting the display height of the track, in pixels. Together, the height and width parameters define the track's display rectangle. The upper-left corner of this rectangle lies at (0,0) in the movie's rectangle. The height and width parameters therefore establish the lower-right corner of the track's display rectangle. If you are creating a track that is not displayed, such as a sound track, set the height and width parameters to 0.

trackVolume

The volume setting of the track as a 16-bit, fixed-point number. The high-order 8 bits specify the integer portion; the low-order 8 bits specify the fractional part. Volume values range from –1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. Set this parameter to `kFullVolume` to play the track at its full, natural volume. Set this parameter to `kNoVolume` to set the volume to 0.

function result The identifier of the new track.

trackVolume Constants

`kFullVolume`

Sets the track to full volume (constant value is 1.0).

`kNoVolume`

Sets the track to no volume (constant value is 0.0).

Discussion

Immediately after creating a new track, you should call `NewTrackMedia` (II–1120) to create a media for the track; a track without a media is of no use. The following code

sample creates a new **sprite** track and media, then calls `BeginMediaEdits` (I–56) to prepare to add samples to the media:

```
// NewMovieTrack coding example
// See “Discovering QuickTime,” page 349
#define kSpriteMediaTimeScale      600

track = NewMovieTrack(movie, ((long)1TrackWidth << 16),
                           ((long)1TrackHeight << 16), 0);
media = NewTrackMedia(track, SpriteMediaType,
                     kSpriteMediaTimeScale, NIL, 0);

FailIfSErr(BeginMediaEdits(media));
```

Special Considerations

When you add a track to a movie, the Movie Toolbox automatically adjusts the display `Rect` (IV–2399) structure of the movie. You may want to detect these changes by calling `GetMovieBox` (I–460) so that you can adjust the size of the movie’s display window.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating Tracks and Media Structures” (V–2726)

Related Java Methods

```
quicktime.std.movies.Movie.newTrack(),
quicktime.std.movies.Movie.addTrack()
```

NewMusicMIDIReadHookUPP

Allocates a **Universal Procedure Pointer** for the `MusicMIDIReadHookProc` (III–2109) callback.

```
MusicMIDIReadHookUPP NewMusicMIDIReadHookUPP (
    MusicMIDIReadHookProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

NewMusicMIDISendUPP

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewMusicMIDIReadHookProc.

Programming Info

C interface file: QuickTimeMusic.h

NewMusicMIDISendUPP

Allocates a **Universal Procedure Pointer** for the MusicMIDISendProc (III–2110) callback.

```
MusicMIDISendUPP NewMusicMIDISendUPP (  
    MusicMIDISendProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewMusicMIDISendProc.

Programming Info

C interface file: QuickTimeMusic.h

NewMusicOfflineDataUPP

Allocates a **Universal Procedure Pointer** for the MusicOfflineDataProc (III–2110) callback.

```
MusicOfflineDataUPP NewMusicOfflineDataUPP (  
    MusicOfflineDataProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewMusicOfflineDataProc.

Programming Info

C interface file: QuickTimeMusic.h

NewPrePrerollCompleteUPP

Allocates a **Universal Procedure Pointer** for the PrePrerollCompleteProc (III–2111) callback.

```
PrePrerollCompleteUPP NewPrePrerollCompleteUPP (
    PrePrerollCompleteProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewPrePrerollCompleteProc.

Programming Info

C interface file: MediaHandlers.h

NewPreprocessInstructionHandlerUPP

Undocumented

```
PreprocessInstructionHandlerUPP NewPreprocessInstructionHandlerUPP (
    PreprocessInstructionHandler    userRoutine );
```

userRoutine

Undocumented

NewQDPixUPP

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

NewQDPixUPP

Allocates a **Universal Procedure Pointer** for the QDPixProc (III–2117) callback.

```
QDPixUPP NewQDPixUPP (  
    QDPixProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewQDPixProc`.

Programming Info

C interface file: `ImageCompression.h`

NewQTBandwidthNotificationUPP

Allocates a **Universal Procedure Pointer** for the QTBandwidthNotificationProc (III–2124) callback.

```
QTBandwidthNotificationUPP NewQTBandwidthNotificationUPP (  
    QTBandwidthNotificationProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewQTBandwidthNotificationProc.

Programming Info

C interface file: `Movies.h`

NewQTCallBackUPP

Allocates a **Universal Procedure Pointer** for the QTCallBackProc (III–2124) callback.

```
QTCallBackUPP NewQTCallBackUPP (
    QTCallBackProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewQTCallBackProc.

Programming Info

C interface file: `Movies.h`

NewQTSMoalFilterUPP

Allocates a **Universal Procedure Pointer** for the QTSMoalFilterProc (III–2125) callback.

```
QTSMoalFilterUPP NewQTSMoalFilterUPP (
    QTSMoalFilterProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

NewQTSNotificationUPP

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

NewQTSNotificationUPP

Allocates a **Universal Procedure Pointer** for the QTSNotificationProc (III–2126) callback.

```
QTSNotificationUPP NewQTSNotificationUPP (  
    QTSNotificationProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewQTSNotificationProc.

Programming Info

C interface file: QuickTimeStreaming.h

NewQTSPanelFilterUPP

Allocates a **Universal Procedure Pointer** for the QTSPanelFilterProc (III–2126) callback.

```
QTSPanelFilterUPP NewQTSPanelFilterUPP (  
    QTSPanelFilterProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

NewQTSyncTaskUPP

Allocates a **Universal Procedure Pointer** for the QTSyncTaskProc (III–2127) callback.

```
QTSyncTaskUPP NewQTSyncTaskUPP (
    QTSyncTaskProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewQTSyncTaskProc.

Programming Info

C interface file: Movies.h

NewQTVRBackBufferImagingUPP

Allocates a **Universal Procedure Pointer** for the QTVRBackBufferImagingProc (III–2127) callback.

```
QTVRBackBufferImagingUPP NewQTVRBackBufferImagingUPP (
    QTVRBackBufferImagingProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewQTVRBackBufferImagingProc.

NewQTVREnteringNodeUPP

Programming Info

C interface file: QuickTimeVR.h

NewQTVREnteringNodeUPP

Allocates a **Universal Procedure Pointer** for the QTVREnteringNodeProc (III–2128) callback.

```
QTVREnteringNodeUPP NewQTVREnteringNodeUPP (  
    QTVREnteringNodeProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewQTVREnteringNodeProc.

Programming Info

C interface file: QuickTimeVR.h

NewQTVRImagingCompleteUPP

Allocates a **Universal Procedure Pointer** for the QTVRImagingCompleteProc (III–2129) callback.

```
QTVRImagingCompleteUPP NewQTVRImagingCompleteUPP (  
    QTVRImagingCompleteProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewQTVRImagingCompleteProc.

Programming Info

C interface file: QuickTimeVR.h

NewQTVRInterceptUPP

Allocates a **Universal Procedure Pointer** for the QTVRInterceptProc (III–2130) callback.

```
QTVRInterceptUPP NewQTVRInterceptUPP (
    QTVRInterceptProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewQTVRInterceptProc.

Programming Info

C interface file: QuickTimeVR.h

NewQTVRLeavingNodeUPP

Allocates a **Universal Procedure Pointer** for the QTVRLeavingNodeProc (III–2130) callback.

```
QTVRLeavingNodeUPP NewQTVRLeavingNodeUPP (
    QTVRLeavingNodeProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

NewQTVRMouseOverHotSpotUPP

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewQTVRLeavingNodeProc.

Programming Info

C interface file: QuickTimeVR.h

NewQTVRMouseOverHotSpotUPP

Allocates a **Universal Procedure Pointer** for the QTVRMouseOverHotSpotProc (III–2131) callback.

```
QTVRMouseOverHotSpotUPP NewQTVRMouseOverHotSpotUPP (  
    QTVRMouseOverHotSpotProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewQTVRMouseOverHotSpotProc.

Programming Info

C interface file: QuickTimeVR.h

NewRTPMPDataReleaseUPP

Allocates a **Universal Procedure Pointer** for the RTPMPDataReleaseProc (III–2132) callback.

```
RTPMPDataReleaseUPP NewRTPMPDataReleaseUPP (  
    RTPMPDataReleaseProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewRTPMPDataReleaseProc.

Programming Info

C interface file: QTStreamingComponents.h

NewRTPPBCallbackUPP

Allocates a **Universal Procedure Pointer** for the RTPPBCallbackProc (III–2133) callback.

```
RTPPBCallbackUPP NewRTPPBCallbackUPP (
    RTPPBCallbackProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewRTPPBCallbackProc.

Programming Info

C interface file: QTStreamingComponents.h

NewSCModalFilterUPP

Allocates a **Universal Procedure Pointer** for the SCModalFilterProc (III–2133) callback.

```
SCModalFilterUPP NewSCModalFilterUPP (
    SCModalFilterProcPtr    userRoutine );
```

NewSCModalHookUPP

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSCModalFilterProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSCModalHookUPP

Allocates a **Universal Procedure Pointer** for the `SCModalHookProc` (III–2134) callback.

```
SCModalHookUPP NewSCModalHookUPP (  
    SCModalHookProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSCModalHookProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSGAddFrameBottleUPP

Allocates a **Universal Procedure Pointer** for the `SGAddFrameBottleProc` (III–2136) callback.

```
SGAddFrameBottleUPP NewSGAddFrameBottleUPP (  
    SGAddFrameBottleProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGAddFrameBottleProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSGCompressBottleUPP

Allocates a **Universal Procedure Pointer** for the `SGCompressBottleProc` (III–2137) callback.

```
SGCompressBottleUPP NewSGCompressBottleUPP (
    SGCompressBottleProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGCompressBottleProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSGCompressCompleteBottleUPP

Allocates a **Universal Procedure Pointer** for the `SGCompressCompleteBottleProc` (III–2138) callback.

```
SGCompressCompleteBottleUPP NewSGCompressCompleteBottleUPP (
    SGCompressCompleteBottleProcPtr    userRoutine );
```

NewSGDataUPP

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGCompressCompleteBottleProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSGDataUPP

Allocates a **Universal Procedure Pointer** for the `SGDataProc` (III–2139) callback.

```
SGDataUPP NewSGDataUPP (  
    SGDataProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGDataProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSGDisplayBottleUPP

Allocates a **Universal Procedure Pointer** for the `SGDisplayBottleProc` (III–2140) callback.

```
SGDisplayBottleUPP NewSGDisplayBottleUPP (  
    SGDisplayBottleProcPtr    userRoutine );
```


`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGDisplayBottleProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSGDisplayCompressBottleUPP

Allocates a **Universal Procedure Pointer** for the `SGDisplayCompressBottleProc` (III–2141) callback.

```
SGDisplayCompressBottleUPP NewSGDisplayCompressBottleUPP (  
    SGDisplayCompressBottleProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGDisplayCompressBottleProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSGGrabBottleUPP

Allocates a **Universal Procedure Pointer** for the `SGGrabBottleProc` (III–2142) callback.

```
SGGrabBottleUPP NewSGGrabBottleUPP (  
    SGGrabBottleProcPtr    userRoutine );
```

NewSGGrabCompleteBottleUPP

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewSGGrabBottleProc.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSGGrabCompleteBottleUPP

Allocates a **Universal Procedure Pointer** for the `SGGrabCompleteBottleProc` (III–2143) callback.

```
SGGrabCompleteBottleUPP NewSGGrabCompleteBottleUPP (  
    SGGrabCompleteBottleProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewSGGrabCompleteBottleProc.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSGGrabCompressCompleteBottleUPP

Allocates a **Universal Procedure Pointer** for the `SGGrabCompressCompleteBottleProc` (III–2143) callback.

```
SGGrabCompressCompleteBottleUPP NewSGGrabCompressCompleteBottleUPP (  
    SGGrabCompressCompleteBottleProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGGrabCompressCompleteBottleProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSGModalFilterUPP

Allocates a **Universal Procedure Pointer** for the `SGModalFilterProc` (III–2144) callback.

```
SGModalFilterUPP NewSGModalFilterUPP (
    SGModalFilterProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGModalFilterProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSGTransferFrameBottleUPP

Allocates a **Universal Procedure Pointer** for the `SGTransferFrameBottleProc` (III–2145) callback.

```
SGTransferFrameBottleUPP NewSGTransferFrameBottleUPP (
    SGTransferFrameBottleProcPtr    userRoutine );
```

NewSICompletionUPP

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGTransferFrameBottleProc`.

Programming Info

C interface file: `QuickTimeComponents.h`

NewSICompletionUPP

Allocates a **Universal Procedure Pointer** for the `SICompletionProc` (III–2146) callback.

```
SICompletionUPP NewSICompletionUPP (  
    SICompletionProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSICompletionProc`.

Programming Info

C interface file: `Sound.h`

NewSIInterruptUPP

Allocates a **Universal Procedure Pointer** for the `SIInterruptProc` (III–2147) callback.

```
SIInterruptUPP NewSIInterruptUPP (  
    SIInterruptProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSIInterruptProc`.

Programming Info

C interface file: `Sound.h`

NewSndCallbackUPP

Allocates a **Universal Procedure Pointer** for the `SndCallbackProc` (III–2147) callback.

```
SndCallbackUPP NewSndCallbackUPP (
    SndCallbackProcPtr    userRoutine );
```

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSndCallbackProc`.

Programming Info

C interface file: `Sound.h`

NewSndDoubleBackUPP

Obsolete.

```
SndDoubleBackUPP NewSndDoubleBackUPP (
    SndDoubleBackProcPtr    userRoutine );
```

NewSoundConverterFillBufferDataUPP

Version Notes

Introduced in QuickTime 4.1 to replace NewSndDoubleBackProc. **Macintosh Note:** Not supported by **CarbonLib**.

Programming Info

C interface file: Sound.h

NewSoundConverterFillBufferDataUPP

Allocates a **Universal Procedure Pointer** for the SoundConverterFillBufferDataProc (III–2148) callback.

```
SoundConverterFillBufferDataUPP NewSoundConverterFillBufferDataUPP (  
    SoundConverterFillBufferDataProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Sound.h

NewSoundParamUPP

Allocates a **Universal Procedure Pointer** for the SoundParamProc (III–2149) callback.

```
SoundParamUPP NewSoundParamUPP (  
    SoundParamProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewSoundParamProc.

Programming Info

C interface file: Sound.h

NewSprite

Creates a new **sprite** in a specified sprite world.

```
OSErr NewSprite (
    Sprite                *newSprite,
    SpriteWorld           itsSpriteWorld,
    ImageDescriptionHandle idh,
    Ptr                   imageDataPtr,
    MatrixRecord          *matrix,
    Boolean               visible,
    short                 layer );
```

newSprite

A pointer to field that is to receive the new **sprite**'s identifier. On return, this field contains the identifier of the newly created sprite.

itsSpriteWorld

The **sprite** world with which the new sprite should be associated.

idh

A handle to an ImageDescription (IV-2285) structure of the **sprite**'s image.

imageDataPtr

A pointer to the **sprite**'s image data.

matrix

A pointer to the **sprite**'s MatrixRecord (IV-2304) structure. If you pass NIL, an identity matrix is assigned to the sprite.

visible

Specifies whether the **sprite** is visible.

layer

The **sprite**'s layer. Sprites with lower layer values appear in front of sprites with higher layer values. If you want to create a sprite that is drawn to the background

NewSpriteWorld

graphics world, you should specify the constant `kBackgroundSpriteLayerNum` for the `layer` parameter.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The visible parameter, the `layer` parameter, and the `newSprite` and `itsSpriteWorld` parameters are required. You can defer assigning image data to the **sprite** by passing `NIL` for both the `idh` and `imageDataPtr` parameters. If you choose to defer assigning image data, you must call `SetSpriteProperty` (III–1641) to assign the image description handle and image data to the sprite before the next call to `SpriteWorldIdle` (III–1908).

Special Considerations

The caller owns the image description handle and the image data pointer; it is the caller’s responsibility to dispose of them after it disposes of a **sprite**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Manipulating Sprites” (V–2901)

Related Java Methods

`quicktime.std.anim.Sprite.Sprite()`

See Also

Once you have created the **sprite**, you can manipulate it using `SetSpriteProperty` (III–1641).

NewSpriteWorld

Creates a new **sprite** world.

```
OSErr NewSpriteWorld (
    SpriteWorld    *newSpriteWorld,
    GWorldPtr      destination,
    GWorldPtr      spriteLayer,
    RGBColor       *backgroundColor,
    GWorldPtr      background );
```


`newSpriteWorld`

A pointer to a field that is to receive the new **sprite** world's identifier. On return, this field contains the identifier for the newly created sprite world.

`destination`

A pointer to a `CGrafPort` (IV–2168) structure that defines the graphics world to be used as the destination.

`spriteLayer`

A pointer to a `CGrafPort` (IV–2168) structure that defines the graphics world to be used as the **sprite** layer.

`backgroundColor`

A pointer to an `RGBColor` (IV–2403) structure that defines the color to be used as the background color. If you pass a background graphics world to this function by setting the background parameter, you can set this parameter to `NIL`.

`background`

A pointer to a `CGrafPort` (IV–2168) structure that defines the graphics world to be used as the background. If you pass a background color to this function by setting the `backgroundColor` parameter, you can set this parameter to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You call this function to create a new **sprite** world with associated destination and sprite layer graphics worlds, and either a background color or a background graphics world. Once created, you can manipulate the sprite world and add sprites to it using other sprite Movie Toolbox functions.

The `newSpriteWorld`, `destination`, and `spriteLayer` parameters are all required. You should specify a background color, a background graphics world, or both. You should not pass `NIL` for both parameters. If you specify both a background graphics world and a background color, the sprite world is filled with the background color before the background sprites are drawn. If no background color is specified, black is the default. If you specify a background graphics world, it should have the same dimensions and depth as the graphics world specified by `spriteLayer`. If you draw to the graphics worlds associated with a sprite world using standard `QuickDraw` and `QuickTime` functions, your drawing is erased by the sprite world's background color. The sprite world created by this function has an identity matrix and does not have a clip shape.

Here is an example of creating a sprite world:

NewSpriteWorld

```
// NewSpriteWorld coding example
// See "Discovering QuickTime," page 166
GWorldPtr      pSpritePlane = NIL;
SpriteWorld    spriteWorld = NIL;
Rect           rectBounce;
RGBColor       rgbcBackground;

void CreateSpriteStuff (Rect *pWndRect, CGrafPtr pMacWnd)
{
    OSErr      nErr;
    Rect       rect;
    // calculate the size of the destination
    rect = *pWndRect;
    OffsetRect(&rect, -rect.left, -rect.top);
    rectBounce = rect;
    InsetRect(&rectBounce, 16, 16);

    // create a sprite graphics world with a bit depth of 16
    NewGWorld(&pSpritePlane, 16, &rect, NIL, NIL, useTempMem);
    if (pSpritePlane == NIL)
        NewGWorld(&pSpritePlane, 16, &rect, NIL, NIL, 0);

    if (pSpritePlane != NIL) {
        LockPixels(pSpritePlane->portPixMap);
        rgbcBackground.red =
        rgbcBackground.green =
        rgbcBackground.blue = 0;

        // create a sprite world
        nErr = NewSpriteWorld(&spriteWorld, (CGrafPtr)pMacWnd,
            pSpritePlane, &rgbBackground, NIL);
    }
}
```

Special Considerations

Before calling this function, you should lock the pixel maps of the **sprite** layer and background graphics worlds. These graphics worlds must remain valid and locked for the lifetime of the sprite world. The sprite world does not own the graphics worlds that are associated with it; it is the caller's responsibility to dispose of the graphics worlds when they are no longer needed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working with Sprite Worlds” (V–2900)

Related Java Methods

`quicktime.std.anim.SpriteWorld.SpriteWorld()`

NewStartDocumentHandlerUPP

Undocumented

```
StartDocumentHandlerUPP NewStartDocumentHandlerUPP (  
    StartDocumentHandler    userRoutine );
```

`userRoutine`

Undocumented

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

NewStartElementHandlerUPP

Undocumented

```
StartElementHandlerUPP NewStartElementHandlerUPP (  
    StartElementHandler    userRoutine );
```

`userRoutine`

Undocumented

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

NewStdPixUPP

NewStdPixUPP

Allocates a **Universal Procedure Pointer** for the StdPixProc (III–2149) callback.

```
StdPixUPP NewStdPixUPP (  
    StdPixProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewStdPixProc.

Programming Info

C interface file: ImageCompression.h

NewTextMediaUPP

Allocates a **Universal Procedure Pointer** for the TextMediaProc (III–2150) callback.

```
TextMediaUPP NewTextMediaUPP (  
    TextMediaProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewTextMediaProc.

Programming Info

C interface file: Movies.h

NewTimeBase

Obtains a new time base.

```
TimeBase NewTimeBase ( void );
```

function result The ID of the new time base.

Discussion

This function sets the rate of the time base to 0, the start time to its minimum value, the time value to 0, and the stop time to its maximum value. The function assigns the default clock component to the new time base.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Disposing of Time Bases” (V–2759)

Related Java Methods

```
quicktime.std.clocks.TimeBase.TimeBase()
```

See Also

If you want to assign a different clock component or a master time base to the new time base, use `SetTimeBaseMasterClock` (III–1649) or `SetTimeBaseMasterTimeBase` (III–1650).

NewTrackEditState

Creates a new edit state for a given track.

```
TrackEditState NewTrackEditState (
    Track    theTrack );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result The track’s edit state identifier. If the edit state could not be created, the returned identifier is set to `NIL`. You must dispose of a movie’s track edit states, using `Use DisposeTrackEditState` (I–300), before disposing of the track or of the movie that contains the track.

NewTrackMedia

Discussion

Use the returned identifier with other Movie Toolbox edit state functions, such as `UseTrackEditState` (III–1984). The edit state contains all the information describing a track’s content, including the identity of the media data associated with the track and all the track’s edit lists.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Undo for Tracks” (V–2751)

Related Java Methods

`quicktime.std.movies.Track.newEditState()`

NewTrackMedia

Creates a media for a new track.

```
Media NewTrackMedia (
    Track          theTrack,
    OSType         mediaType,
    TimeScale      timeScale,
    Handle         dataRef,
    OSType         dataRefType );
```

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092).

`mediaType`

The type of media to create; see “Codec Identifiers” (IV–2655). The Movie Toolbox uses this value to find the correct media handler for the new media. If the Movie Toolbox cannot locate an appropriate media handler, it returns an error.

`timeScale`

Defines the media’s time coordinate system.

`dataRef`

The data reference. This parameter contains a handle to the information that identifies the file that contains this media’s data. The type of information stored in that handle depends upon the value of the `dataRefType` parameter. If you are creating a new media that refers to existing media data, you can use the

GetMediaDataRef (I–427) function to obtain information about the existing data reference. You can then supply information about that reference to this function. Set this parameter to NIL to use the file that is associated with the movie or if the movie does not have a movie file. For example, if you have created the movie using CreateMovieFile (I–145) or NewMovieFromFile (II–1080), the Movie Toolbox assumes that the movie’s data resides in the file specified at that time. If you have created the movie using the NewMovieFromScrap (II–1085) or NewMovie (II–1069) functions, the movie does not have a movie file.

dataRefType

The type of data reference; see “Data References” (IV–2661). If the data reference is an **alias**, you must set this parameter to rAliasType. See *Inside Macintosh: Files* (listed in the **bibliography**) for more information about aliases and the Alias Manager.

function result A media identifier, referring to the actual data samples used by the track. If the function cannot create a new media, it sets the returned value to NIL.

Discussion

The following code sample creates a new **sprite** track and media, then calls BeginMediaEdits (I–56) to prepare to add samples to the media:

```
// NewTrackMedia coding example
// See “Discovering QuickTime,” page 349
#define kSpriteMediaTimeScale      600

track = NewMovieTrack(movie, ((long)lTrackWidth << 16),
                           ((long)lTrackHeight << 16), 0);
media = NewTrackMedia(track, SpriteMediaType,
                     kSpriteMediaTimeScale, NIL, 0);

FailOSErr(BeginMediaEdits(media));
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Creating Tracks and Media Structures” (V–2726)

Related Java Methods

```
quicktime.std.movies.media.MusicMedia.MusicMedia(),
quicktime.std.movies.media.TimeCodeMedia.TimeCodeMedia(),
quicktime.std.movies.media.TextMedia.TextMedia(),
```

NewTrackTransferUPP

```
quicktime.std.movies.media.VideoMedia.VideoMedia(),
quicktime.std.movies.media.BaseMedia.BaseMedia(),
quicktime.std.movies.media.TweenMedia.TweenMedia(),
quicktime.std.movies.media.SpriteMedia.SpriteMedia(),
quicktime.std.movies.media.MPEGMedia.MPEGMedia(),
quicktime.std.movies.media.SoundMedia.SoundMedia(),
quicktime.std.movies.media.Media.newFromType(),
quicktime.std.movies.media.ThreeDMedia.ThreeDMedia()
```

NewTrackTransferUPP

Allocates a **Universal Procedure Pointer** for the TrackTransferProc (III–2151) callback.

```
TrackTransferUPP NewTrackTransferUPP (
    TrackTransferProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewTrackTransferProc.

Programming Info

C interface file: Movies.h

NewTuneCallbackUPP

Allocates a **Universal Procedure Pointer** for the TuneCallbackProc (III–2152) callback.

```
TuneCallbackUPP NewTuneCallbackUPP (
    TuneCallbackProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewTuneCallBackProc.

Programming Info

C interface file: QuickTimeMusic.h

NewTunePlayCallBackUPP

Allocates a **Universal Procedure Pointer** for the TunePlayCallBackProc (III–2152) callback.

```
TunePlayCallBackUPP NewTunePlayCallBackUPP (
    TunePlayCallBackProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewTunePlayCallBackProc.

Programming Info

C interface file: QuickTimeMusic.h

NewTweenDataUPP

Allocates a **Universal Procedure Pointer** for the TweenerDataProc (III–2153) callback.

```
TweenerDataUPP NewTweenDataUPP (
    TweenerDataProcPtr    userRoutine );
```

NewUserData

`userRoutine`

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces `NewTweenerDataProc`.

Programming Info

C interface file: `Movies.h`

NewUserData

Creates a new user data structure.

```
OSErr NewUserData (
    UserData    *theUserData );
```

`theUserData`

A pointer to a pointer to a new `UserDataRecord` (IV–2496) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. If the function fails, `theUserData` is set to `NIL`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

`quicktime.std.movies.media.UserData.UserData()`

NewUserDataFromHandle

Creates a new user data structure from a handle.

```
OSErr NewUserDataFromHandle (
    Handle      h,
    UserData    *theUserData );
```

h

A handle to the data structure specified in theUserData.

theUserData

A pointer to a pointer to a new UserDataRecord (IV–2496) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error. If the function fails, theUserData is set to NIL.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

quicktime.std.movies.media.UserData.UserData()

NewVdigIntUPP

Allocates a **Universal Procedure Pointer** for the VdigIntProc (III–2154) callback.

```
VdigIntUPP NewVdigIntUPP (
    VdigIntProcPtr    userRoutine );
```

userRoutine

Your application-defined function.

function result A new UPP; see “Universal Procedure Pointers” (IV–2641).

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 4.1. Replaces NewVdigIntProc.

Programming Info

C interface file: QuickTimeComponents.h

O

OpenAComponent

Gains access to the services provided by a component specified by a component identifier and returns an `OSErr` value.

```
OSErr OpenAComponent (
    Component      aComponent,
    ComponentInstance *ci );
```

aComponent

A component identifier that specifies the component to open. Your application obtains this identifier from `FindNextComponent` (I–360). If your application registers a component, it can also obtain a component identifier from `RegisterComponent` (III–1451) or `RegisterComponentResource` (III–1453).

ci

A pointer to a field to receive the instance of the newly-opened component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Component Functions Used By Applications” (V–2781)

Related Java Methods

```
quicktime.std.comp.Component.Component(),
quicktime.std.qtcomponents.DataCodecDecompressor.DataCodecDecompressor(),
quicktime.std.qtcomponents.MovieExporter.MovieExporter(),
quicktime.std.qtcomponents.MovieImporter.MovieImporter(),
quicktime.std.qtcomponents.DataCodecCompressor.DataCodecCompressor()
```

See Also

This function acts similarly to `OpenComponent` (II–1130), except that it provides an `OSErr` value return.

OpenAComponentResFile

Allows your component to gain access to its **resource** file and returns an OSErr value.

```
OSErr OpenAComponentResFile (
    Component    aComponent,
    short        *resRef );
```

aComponent

A component identifier that specifies the component whose **resource** file you want to open. Your application can obtain this identifier from RegisterComponentResource (III–1453) or FindNextComponent (I–360); or it can be a ComponentInstance value, in which case it's returned by OpenAComponent (II–1126) or OpenADefaultComponent (II–1129).

resRef

A pointer to a field to receive the **resource** reference number of the newly opened component resource file.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Components.h

Programming summary: “Functions Used By Components” (V–2782)

See Also

This function acts similarly to OpenComponentResFile (II–1130), except that it provides an OSErr value return.

OpenADataHandler

Opens a data handler component.

```
OSErr OpenADataHandler (
    Handle        dataRef,
    OSType        dataHandlerSubType,
    Handle        anchorDataRef,
    OSType        anchorDataRefType,
    TimeBase      tb,
    long          flags,
    ComponentInstance *dh );
```



OpenADataHandler

`dataRef`

A handle to a data reference. The type of information stored in the handle depends upon the data reference type specified by the `dataHandlerSubType` parameter.

`dataHandlerSubType`

Identifies both the type of data reference and, by implication, the component subtype value assigned to the data handler components that operate on data references of that type.

`anchorDataRef`

A handle to the anchor data reference.

`anchorDataRefType`

The type of the anchor data reference.

`tb`

The time base for the data handler. Your application obtains this time base identifier from `NewTimeBase` (II–1119).

`flags`

Flags (see below) that indicate the way in which you intend to use the data handler component. Not all data handlers necessarily support all services; for example, some data handler components may not support streaming writes. Set the appropriate flags to 1.

`dh`

A pointer to a field to receive the `ComponentInstance` value of the newly-opened data handler component.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

flags Constants

`kDataHCanRead`

You intend to use the data handler component to read data.

`kDataHCanWrite`

You intend to use the data handler component to write data.

`kDataHCanStreamingWrite`

You intend to do streaming writes (as part of a movie-capture operation, for example).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

OpenADefaultComponent

Gains access to the services provided by a component, specified by component type and subtype, and returns an `OSErr` value.

```
OSErr OpenADefaultComponent (
    OSType          componentType,
    OSType          componentSubType,
    ComponentInstance *ci );
```

`componentType`

A four-character code that identifies the type of component. All components of a particular type support a common set of interface routines. Your application uses this field to search for components of a given type.

`componentSubType`

A four-character code that identifies the subtype of the component. Different subtypes of a component type may support additional features or provide interfaces that extend beyond the standard routines for a given component type. For example, the subtype of an image compressor component indicates the compression algorithm employed by the compressor. Your application can use the `componentSubType` field to perform a more specific lookup operation than is possible using only the `componentType` field. For example, you may want your application to use only components of a certain component type ('draw') that also have a specific subtype ('oval'). Set this parameter to 0 to select a component with any subtype value.

`ci`

A pointer to a field to receive the `ComponentInstance` value of the newly-opened component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Component Functions Used By Applications” (V–2781)

Related Java Methods

`quicktime.std.image.GraphicsImporter.GraphicsImporter()`,

OpenComponent

```
quicktime.std.qtcomponents.DataCodecDecompressor.DataCodecDecompressor(),
quicktime.std.comp.Component.Component(),
quicktime.std.qtcomponents.MovieExporter.MovieExporter(),
quicktime.std.qtcomponents.MovieImporter.MovieImporter(),
quicktime.std.music.MusicComponent.MusicComponent(),
quicktime.std.clocks.Clock.Clock(),
quicktime.std.music.NoteAllocator.NoteAllocator(),
quicktime.std.music.TunePlayer.TunePlayer(),
quicktime.std.qtcomponents.DataCodecCompressor.DataCodecCompressor(),
quicktime.std.sg.SequenceGrabber.SequenceGrabber(),
quicktime.std.sg.VideoDigitizer.VideoDigitizer()
```

See Also

This function acts similarly to `OpenDefaultComponent` (II–1131), except that its return value is an `OSErr` value.

OpenComponent

Gains access to the services provided by a component specified by a component identifier.

```
ComponentInstance OpenComponent (
    Component      aComponent );
```

`aComponent`

A component identifier that specifies the component to open. Your application obtains this identifier from `FindNextComponent` (I–360). If your application registers a component, it can also obtain a component identifier from `RegisterComponent` (III–1451) or `RegisterComponentResource` (III–1453).

function result An instance of the component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Component Functions Used By Applications” (V–2781)

OpenComponentResFile

Allows your component to gain access to its **resource** file.


```
short GetComponentResFile (
    Component    aComponent );
```

aComponent

A component identifier that specifies the component whose **resource** file you want to open. Your application can obtain this identifier from `RegisterComponentResource` (III–1453) or `FindNextComponent` (I–360); or it can be a `ComponentInstance` value, in which case it's returned by `OpenAComponent` (II–1126) or `OpenADefaultComponent` (II–1129).

function result The **resource** reference number of the newly opened component resource file.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

OpenDefaultComponent

Gains access to the services provided by a component specified by component type and subtype.

```
ComponentInstance OpenDefaultComponent (
    OSType    componentType,
    OSType    componentSubType );
```

componentType

A four-character code that identifies the type of component. All components of a particular type support a common set of interface routines. Your application uses this field to search for components of a given type.

componentSubType

A four-character code that identifies the subtype of the component. Different subtypes of a component type may support additional features or provide interfaces that extend beyond the standard routines for a given component type. For example, the subtype of an image compressor component indicates the compression algorithm employed by the compressor. Your application can use the `componentSubType` field to perform a more specific lookup operation than is possible using only the `componentType` field. For example, you may want your application to use only components of a certain component type ('draw')

OpenMixerSoundComponent

that also have a specific subtype ('oval'). Set this parameter to 0 to select a component with any subtype value.

function result The ComponentInstance value of the newly-opened component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Components.h

Programming summary: “Component Functions Used By Applications” (V-2781)

OpenMixerSoundComponent

Opens the Apple Mixer sound component.

```
OSErr OpenMixerSoundComponent (
    SoundComponentDataPtr  outputDescription,
    long                   outputFlags,
    ComponentInstance       *mixerComponent );
```

outputDescription

A pointer to a SoundComponentData (IV-2448) structure for the mixer output.

outputFlags

Flags (see below) that provide information about the sound component chain.

mixerComponent

A pointer to memory that is to receive an instance of the Apple Mixer component.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

outputFlags Constants

kNoMixing

The Apple Mixer does not mix audio data sources.

kNoSampleRateConversion

The sound component chain does not perform sample rate conversion (for example, from 11 kHz data to 22 kHz data).

kNoSampleSizeConversion

The sound component chain does not perform sample size conversion (for example, from 8-bit data to 16-bit data).

kNoSampleFormatConversion

The sound component chain does not convert between sample formats (for example, converting from two's complement data to offset binary data). Most sound output devices on Macintosh computers accept only 8-bit offset binary data, which is therefore the default type of data produced by the Apple Mixer. If your output device can handle either offset binary or two's complement data, you should set this flag. Note that 16-bit data is always in two's complement format.

kNoChannelConversion

The sound component chain does not convert channels (for example, between monophonic and stereo).

kNoDecompression

The sound component chain does not decompress audio data. If your output device can decompress data, you should set this flag.

kNoVolumeConversion

The sound component chain does not convert volumes.

kNoRealtimeProcessing

The sound component chain does not do any processing at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

OpenMovieFile

Opens a specified movie file.

```
OSErr OpenMovieFile (
    const FSSpec    *fileSpec,
    short           *resRefNum,
    SInt8           permission );
```

fileSpec

A pointer to the `FSSpec` (IV–2262) structure for the movie file to be opened.

resRefNum

A pointer to a field that is to receive the file reference number for the opened movie file. Your application must use this value when calling other Movie Toolbox functions that work with movie files. This reference number refers to

OpenMovieFile

the file fork that contains the movie **resource**. If the movie is stored in the data fork of the file, the returned reference number corresponds to the data fork.

permission

The permission level for the file (see below). If your application is only going to play the movie that is stored in the file, you can open the file with read permission. If you plan to add data to the file or change data in the file, you should open the file with write permission.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

permission Constants

`fsCurPerm`

Whatever permission is allowed.

`fsRdPerm`

Read permission.

`fsWrPerm`

Write permission.

`fsRdWrPerm`

Exclusive read / write permission.

`fsRdWrShPerm`

Shared read / write permission.

Discussion

Your application must open a movie file before reading movie data from it or writing movie data to it. You can open a movie file more than once; be sure to call `CloseMovieFile` (I-102) once for each time you call this function. Note that opening the movie file with write permission does not prevent other applications from reading data from the movie file.

If the specified file has a **resource** fork, this function opens the resource fork and returns a file reference number to the resource fork. If the movie file does not have a resource fork (that is, it is a single-fork movie file), this function opens the data fork instead. In this case, your application cannot use `AddMovieResource` (I-36) with the movie file.

The following is an example of using `OpenMovieFile`:

```
// OpenMovieFile coding example
// See “Discovering QuickTime,” page 385
```

```

Movie MyGetMovie (void)
{
    OSErr          nErr;
    SFTypelist      types = {MovieFileType, 0, 0, 0};
    StandardFileReply sfr;
    Movie           movie = NIL;
    short           nFileRefNum;

    StandardGetFilePreview(NIL, 1, types, &sfr);
    if (sfr.sfGood) {
        nErr = OpenMovieFile(&sfr.sfFile, &nFileRefNum, fsRdPerm);
        if (nErr == noErr) {
            short          nResID = 0;          //We want the first movie.
            Str255          strName;
            Boolean         bWasChanged;

            nErr = NewMovieFromFile(&movie, nFileRefNum, &nResID, strName,
                                   newMovieActive, &bWasChanged);
            CloseMovieFile(nFileRefNum);
        }
    }
    return movie;
}

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Functions” (V-2713)

Related Java Methods

`quicktime.io.OpenMovieFile.asRead()`,
`quicktime.io.OpenMovieFile.asWrite()`



ParseAIFFHeader

P

ParseAIFFHeader

Parses out the header in an AIFF file.

```
OSErr ParseAIFFHeader (
    short          fRefNum,
    SoundComponentData *sndInfo,
    unsigned long   *numFrames,
    unsigned long   *dataOffset );
```

fRefNum

A file reference number for an AIFF file.

sndInfo

Pointer to a SoundComponentData (IV–2448) structure in which information is returned.

numFrames

Returns the number of frames of data.

dataOffset

Returns the byte offset of the first data sample.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

Related Java Methods

quicktime.sound.SndInfo.parseAIFFHeader()

ParseSndHeader

Returns information describing the audio data in a given 'snd ' **resource** handle.

```
OSErr ParseSndHeader (
    SndListHandle    sndHandle,
    SoundComponentData *sndInfo,
    unsigned long     *numFrames,
```

```
unsigned long          *dataOffset );
```

sndHandle

The sound handle to use.

sndInfo

Pointer to a `SoundComponentData` (IV-2448) structure in which information is returned.

numFrames

Returns the number of frames of audio data in the handle.

dataOffset

Returns the byte offset of the first audio sample in the handle.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

Related Java Methods

`quicktime.sound.SndHandle.parseSndHeader()`

PasteHandleIntoMovie

Takes the contents of a specified handle, together with its type, and pastes it into a specified movie.

```
OSErr PasteHandleIntoMovie (
    Handle          h,
    OSType          handleType,
    Movie           theMovie,
    long            flags,
    ComponentInstance userComp );
```

h

The handle to be pasted into the movie indicated by the `theMovie` parameter.

handleType

The data type of the handle specified in the `h` parameter. If the handle is set to 0, the function searches the scrap for a field of the type `handleType`. If both the `h` parameter and the `handleType` parameter are `NIL`, the function uses the first available data from the scrap.

PasteHandleIntoMovie

theMovie

The destination movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

flags

A flag (see below) that can further refine conditions of the paste operation.

userComp

The component or an instance of the component that is to perform the conversion of the data into a QuickTime movie. If you want a particular movie import component to perform the conversion, you may pass the component or an instance of that component. Otherwise, set this parameter to 0 to allow the Movie Toolbox to determine the appropriate component. If you pass in a component instance, this function uses it. This allows you to communicate directly with the component before using this function to establish any conversion parameters. If you pass in a component ID, an instance is created and closed within this function.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

flags Constants

pasteInParallel

Changes the function so that it takes the contents of the specified handle along with its type and adds (rather than inserts) it to the specified movie in an operation analogous to that of `AddMovieSelection` (I–38). This operation does not affect the duration of existing tracks. It does not necessarily create a new track; rather, it uses a piece of an existing track, if possible.

Discussion

If you are just pasting in data from the scrap, it is best to allow this function to retrieve the data from the scrap, rather than doing it yourself. In this way, the function is able to obtain supplemental data from the scrap, if necessary (for example, 'styl' **resources** for 'TEXT'). This function can paste into the current selection in two different ways. If the selection is empty (for example, duration = 0), it adds the data with the appropriate duration. If the selection is not empty, the data is added and then scaled to fit into the duration of the selection. The current selection is deleted, unless you set the *pasteInParallel* flag.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Movie Editing Functions” (V–2746)

Related Java Methods

`quicktime.std.movies.Movie.pasteHandle()`

PasteMovieSelection

Places the tracks from one movie into another movie.

```
void PasteMovieSelection (
    Movie    theMovie,
    Movie    src );
```

theMovie

The destination movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

src

The source movie for this operation. `PasteMovieSelection` places the tracks from this movie in the destination movie.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

Whenever possible, the Movie Toolbox uses existing tracks to store the data to be pasted. Before adding a track to the destination movie, the Toolbox looks in the destination movie for tracks that have the same characteristics as the tracks in the source movie. It considers several characteristics when searching for an appropriate track, including track spatial dimensions, track matrix, track clipping region, track matte, alternate group affiliation, media time scale, media type, media language, and data reference (that is, the two tracks must refer to the same file). If the Movie Toolbox cannot find an appropriate track in the destination movie, it creates a track with the proper characteristics. It removes any empty tracks from the destination movie after the paste operation.

Special Considerations

If you have assigned a **progress function** to the destination movie, the Movie Toolbox calls that progress function during long paste operations.

PlayMoviePreview

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Movie Editing Functions” (V–2746)

Related Java Methods

`quicktime.std.movies.Movie.pasteSelection()`

PlayMoviePreview

Plays a movie’s **preview**.

```
void PlayMoviePreview (
    Movie          theMovie,
    MoviePreviewCallOutUPP  callOutProc,
    long           refcon );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

callOutProc

A pointer to a `MoviePreviewCallOutProc` (III–2105) callback in your application. The Movie Toolbox calls this function repeatedly while the movie **preview** is playing. You can use this function to stop the preview. If you don’t want to assign a function, set this parameter to `NIL`.

refcon

A reference constant that the Movie Toolbox passes to your callback. Use this parameter to point to a data structure containing any information your callback needs.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

This function sets the movie into **preview** mode, plays the movie preview, sets the movie back to normal playback mode, and returns to your application. The Movie Toolbox plays the preview in the movie’s graphics world. Note that if you call the `GetMovieActiveSegment` (I–457) function from within your movie callout function, the Movie Toolbox will have changed the active movie segment to be the preview

segment of the movie. The Movie Toolbox restores the active segment when the preview is done playing.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V-2718)

Related Java Methods

`quicktime.std.movies.Movie.playPreview()`

PrePrerollMovie

Sets up any necessary network connections to receive streaming content.

```
OSErr PrePrerollMovie (
    Movie          m,
    TimeValue      time,
    Fixed          rate,
    MoviePrePrerollCompleteUPP proc,
    void           *refcon );
```

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), and `NewMovieFromHandle` (II-1084).

time

The starting time of the movie segment to play.

rate

The rate at which you anticipate playing the movie. You specify the movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

proc

The `MoviePrePrerollCompleteProc` (III-2104) callback you want called when pre-prerolling is complete. If a completion proc is specified, `PrePrerollMovie` operates asynchronously. You must call `MoviesTask` periodically during asynchronous operation. If no completion proc is specified, `PrePrerollMovie` operates synchronously.

PrerollMovie

refcon

A reference constant that is passed to your callback. Use this parameter to point to a data structure containing any information your callback needs.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

Before a movie is played, it is normally prerolled. During preroll, the Movie Toolbox tells the appropriate media handlers to load the movie data, allocate sound channels, start up image-decompression sequences, and so on. Before a movie that contains streaming content is prerolled, it must be pre-prerolled. This sets up any necessary network connections between the client and the server. If a movie contains streaming content (one or more 'strm' tracks), you must call this function before calling `PrerollMovie` (II-1142). If the movie does not contain streaming content, calling this function has no effect. If your application calls `PrerollMovie`, it should always call this function first. If you play movies using a movie controller, you don't need to preroll or pre-preroll the movie explicitly; it is done for you automatically. If a completion proc is specified in the *proc* parameter, this function operates asynchronously; it returns almost immediately and calls the completion proc when pre-prerolling is complete.

Special Considerations

You must call `MoviesTask` (II-973) periodically to grant time for pre-prerolling during asynchronous operation. If no completion proc is specified, this function operates synchronously; the function will not return until pre-prerolling is complete. This can take a long time, particularly if a dial-up network connection must be established.

Version Notes

Introduced in QuickTime 4. Beginning with QuickTime 4, your application should call this function any time it calls `PrerollMovie` (II-1142).

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V-2722)

Related Java Methods

`quicktime.std.movies.Movie.prePreroll()`

PrerollMovie

Prepares a portion of a movie for playback.

```
OSErr PrerollMovie (
    Movie          theMovie,
    TimeValue      time,
    Fixed          Rate );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

time

The starting time of the movie segment to play.

Rate

The rate at which you anticipate playing the movie. You specify the movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

When your application calls `PrerollMovie`, the Movie Toolbox tells the appropriate media handlers to prepare to play the movie. The media handlers may then load the movie data and perform any other necessary preparations to play the movie, such as allocating sound channels and starting up image-decompression sequences. In this manner, you can eliminate playback stutter when the movie starts playing.

If your application uses QuickTime’s Movie Toolbox to play back movies, there are two choices for how to preroll the movie. Like the movie controller, the Movie Toolbox provides a single function call, `StartMovie` (III–1911), which will both preroll the movie and start it playing. Unlike the movie controller, the Movie Toolbox function doesn’t allow you to specify the rate to play the movie at, but instead assumes the movie’s preferred rate.

Calling `StartMovie`, just like the movie controller’s preroll and play action, first prerolls the movie and then sets it playing. If your application requires more control, the Movie Toolbox provides lower level functions that give you more control:

```
// PrerollMovie coding example
StartMovie(theMovie);
TimeValue timeNow;
```

PreviewEvent

```
Fixed playRate;

timeNow = GetMovieTime(theMovie, NIL);
playRate = GetMoviePreferredRate(theMovie);

PrerollMovie(theMovie, timeNow, playRate);
SetMovieRate(theMovie, playRate);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V–2722)

Related Java Methods

`quicktime.std.movies.Movie.preroll()`

PreviewEvent

May be called as appropriate if a **preview** component handles events.

```
ComponentResult PreviewEvent (
    pnotComponent    p,
    EventRecord       *e,
    Boolean           *handledEvent );
```

p

Specifies your **preview** component. You obtain this identifier from `OpenComponent` (II–1130).

e

A pointer to the event structure for this operation.

handledEvent

A pointer to a Boolean value. If you completely handle an event such as a mouse-down event or keystroke, you should set the `handledEvent` parameter to TRUE. Otherwise, set it to FALSE.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Handling Preview Events” (V–2872)

PreviewMakePreview

Creates **previews** by allocating a handle to data that is to be added to a file.

```
ComponentResult PreviewMakePreview (
    pnotComponent          p,
    OSType                  *previewType,
    Handle                   *previewResult,
    const FSSpec             *sourceFile,
    ICMProgressProcRecordPtr progress );
```

p

Specifies your **preview** component. You obtain this identifier from `OpenComponent` (II–1130).

previewType

A pointer to the type of **preview** component that should be used to display the preview.

previewResult

A pointer to a handle of cached **preview** data created by this function.

sourceFile

A pointer to a reference to the file for which the **preview** is created.

progress

A pointer to an `ICMProgressProcRecord` (IV–2284) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Creating Previews” (V–2873)

PreviewMakePreviewReference

Returns the type and identification number of a **resource** within a file to be used as the **preview** for a file.

PreviewShowData

```
ComponentResult PreviewMakePreviewReference (
    pnotComponent    p,
    OSType            *previewType,
    short             *resID,
    const FSSpec      *sourceFile );
```

p

Specifies your **preview** component. You obtain this identifier from `OpenComponent` (II–1130).

previewType

A pointer to the type of **preview** component that should be used to display the preview.

resID

A pointer to the identification number of a **resource** within the file to be used as the **preview** for the file.

sourceFile

A pointer to an `FSSpec` (IV–2262) structure that provides a reference to the file for which the **preview** is created.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Creating Previews” (V–2873)

PreviewShowData

Displays a **preview** if it does not handle events.

```
ComponentResult PreviewShowData (
    pnotComponent    p,
    OSType            dataType,
    Handle            data,
    const Rect        *inHere );
```

p

Specifies your **preview** component. You obtain this identifier from `OpenComponent` (II–1130).

dataType

The type of handle pointing to the data to be displayed in the **preview**.

data

A handle to the data, which is typically the same as the subtype of your **preview** component.

inHere

A pointer to a Rect (IV–2399) structure that defines the area into which you draw the **preview**. The current port is set to the correct graphics port for drawing. You must not draw outside the given rectangle.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Displaying Previews” (V–2872)

PtInDSequenceData

Tests to see if a compressed image contains data at a given point.

```
OSErr PtInDSequenceData (
    ImageSequence    seqID,
    void             *data,
    Size             dataSize,
    Point            where,
    Boolean          *hit );
```

seqID

The unique sequence identifier that was returned by the DecompressSequenceBegin (I–238) function.

data

Pointer to compressed data in the format specified by the desc param.

dataSize

Size of the compressed data referred to by the data param.

where

A QuickDraw Point (IV–2339) structure of value (0,0), based at the top-left corner of the image.

PtInMovie

hit

A pointer to a field to receive the `Boolean` indicating whether or not the image contained data at the specified point. The `Boolean` will be set to `TRUE` if the point specified by the `where` parameter is contained within the compressed image data specified by the `data param`, or `FALSE` if the specified point falls within a blank portion of the image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

`PtInDSequenceData` allows the application to perform hit testing on compressed data.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Decompression Parameters” (V–2795)

Related Java Methods

`quicktime.std.image.DSequence.ptInData()`

PtInMovie

Determines whether a specified point lies in the region defined by a movie’s final display boundary region after it has been clipped by the movie’s display clipping region.

```
Boolean PtInMovie (  
    Movie    theMovie,  
    Point    pt );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

pt

The point to be checked. This point must be expressed in the movie’s local display coordinate system.

function result Returns `TRUE` if the point is in the movie.

Discussion

This function is accurate at the current movie time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movies and Your Event Loop” (V-2720)

Related Java Methods

`quicktime.std.movies.Movie.pointInMovie()`

PtInTrack

Determines whether a specified point lies in the region defined by a track’s display boundary region after it has been clipped by the movie’s final display clipping region.

```
Boolean PtInTrack (
    Track    theTrack,
    Point    pt );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II-1092) and `GetMovieTrack` (I-497).

pt

The point to be checked. This point must be expressed in the local display coordinate system of the movie that contains the track.

function result Returns `TRUE` if the point lies in the track’s display space.

Discussion

This function is accurate at the current movie time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movies and Your Event Loop” (V-2720)

Related Java Methods

`quicktime.std.movies.Track.pointInMovie()`

PutMovieIntoDataFork

PutMovieIntoDataFork

Stores a movie in the data fork of a given file.

```
OSErr PutMovieIntoDataFork (
    Movie    theMovie,
    short    fRefNum,
    long     offset,
    long     maxSize );
```

theMovie

The movie to be stored in the data fork of an atom. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

fRefNum

A file reference number for the data fork of the given file. You pass in an open write path in the `fRefNum` parameter.

offset

Indicates where the movie should be written.

maxSize

The largest number of bytes that may be written.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Saving Movies” (V–2715)

Related Java Methods

`quicktime.std.movies.Movie.putIntoDataFork()`

PutMovieIntoDataFork64

Provides a 64-bit version of `PutMovieIntoDataFork` (II–1150).

```
OSErr PutMovieIntoDataFork64 (
    Movie    theMovie,
    long     fRefNum,
```

```
const wide      *offset,
unsigned long    maxSize );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

fRefNum

A file reference number for the data fork of the given file. You pass in an open write path in the `fRefNum` parameter.

offset

Pointer to a 64-bit value that indicates where the movie should be written.

maxSize

The largest number of bytes that may be written.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

PutMovieIntoHandle

Creates a new movie **resource**.

```
OSErr PutMovieIntoHandle (
    Movie    theMovie,
    Handle   publicMovie );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

publicMovie

The handle that is to receive the new movie **resource**. The function resizes the handle if necessary.

function result You can access Movie Toolbox error returns through `GetMoviesError`

PutMovieIntoTypedHandle

(I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Use this handle to store a QuickTime movie in a specialized storage format.

Special Considerations

Note that you cannot use this new movie with other Movie Toolbox functions, except for `NewMovieFromHandle` (II–1084).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Saving Movies” (V–2715)

Related Java Methods

`quicktime.std.movies.Movie.putIntoHandle()`

See Also

Use `NewMovieFromHandle` (II–1084) to load a movie from a handle.

PutMovieIntoTypedHandle

Takes a movie, or a single track from within that movie, and converts it into a handle of a specified type.

```
OSErr PutMovieIntoTypedHandle (
    Movie          theMovie,
    Track          targetTrack,
    OSType         handleType,
    Handle         publicMovie,
    TimeValue      start,
    TimeValue      dur,
    long           flags,
    ComponentInstance userComp );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

`targetTrack`

The track to convert.

`handleType`

The type of the new data.

`publicMovie`

The actual handle in which to place the new data.

`start`

The start time of the segment of the movie or track to be converted.

`dur`

The duration of the segment of the movie or track to be converted.

`flags`

Condition of the conversion. Set this parameter to 0.

`userComp`

Indicates a component or component instance of the movie export component you want to perform the conversion. Otherwise, set this parameter to 0 for the Movie Toolbox to choose the appropriate component. If you pass in a component instance, this function will use it. This allows you to communicate directly with the component before using this function to establish any conversion parameters. If you pass in a component ID, an instance is created and closed within this function.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Movie Editing Functions” (V–2746)

Related Java Methods

`quicktime.std.movies.Movie.putIntoTypedHandle()`

PutMovieOnScrap

Places a movie into the Macintosh scrap.

```
OSErr PutMovieOnScrap (
    Movie    theMovie,
    long     movieScrapFlags );
```

PutUserDataIntoHandle

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

movieScrapFlags

Flags (see below) that control the operation. Be sure to set unused flags to 0.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

movieScrapFlags Constants

movieScrapDontZeroScrap

Controls whether the Movie Toolbox clears the scrap before putting the movie on the scrap. If you set this flag to 1, the Movie Toolbox does not clear the scrap before placing your movie onto this scrap, thus adding your movie to the previous contents of the scrap. If you set this flag to 0, the function clears the scrap, then places your movie on the scrap.

movieScrapOnlyPutMovie

Controls whether the Movie Toolbox places other items on the scrap along with your movie. If you set this flag to 1, the Movie Toolbox only places your movie on the scrap. If you set this flag to 0, the Movie Toolbox places an image from the current movie time (including but not limited to a PICT) on the scrap along with your movie. The picture is intended for use by applications that cannot work with movies.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Movie Editing Functions” (V–2746)

Related Java Methods

`quicktime.std.movies.Movie.putOnScrap()`

PutUserDataIntoHandle

Takes a specified user data structure and replaces the contents of the handle with a publicly parsable form of the user data.

```
OSErr PutUserDataIntoHandle (  
    UserData    theUserData,
```



```
Handle h );
```

theUserData

The user data structure that is to be disposed of.

h

A handle to the user data structure specified in the theUserData parameter.

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Working With Movie User Data” (V-2743)

Related Java Methods

quicktime.std.movies.media.UserData.putIntoHandle()

Q

QTBandwidthRelease

Undocumented

```
OSErr QTBandwidthRelease (
    QTBandwidthReference bwRef,
    long flags );
```

bwRef

Undocumented

flags

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

QTBandwidthRequest

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

QTBandwidthRequest

Undocumented

```
OSErr QTBandwidthRequest (
    long                priority,
    QTBandwidthNotificationUPP callback,
    const void          *refcon,
    QTBandwidthReference *bwRef,
    long                flags );
```

priority

Undocumented

callback

A `QTBandwidthNotificationProc` (III–2124) callback.

refcon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

bwRef

Undocumented

flags

Undocumented

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

QTBandwidthRequestForTimeBase

Undocumented

```
OSErr QTBandwidthRequestForTimeBase (
    TimeBase          tb,
    long              priority,
    QTBandwidthNotificationUPP callback,
    const void        *refcon,
    QTBandwidthReference *bwRef,
    long              flags );
```

tb

A time base. Your application obtains this time base identifier from NewTimeBase (II-1119).

priority

Undocumented

callback

A QTBandwidthNotificationProc (III-2124) callback.

refcon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

bwRef

Undocumented

flags

Flags (see below).

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.1.



QTCopyAtom

Programming Info

C interface file: `Movies.h`

QTCopyAtom

Copies an atom and its children to a new atom container.

```
OSErr QTCopyAtom (
    QTAtomContainer    container,
    QTAtom              atom,
    QTAtomContainer     *targetContainer );
```

container

The atom container that contains the atom to be copied.

atom

The atom to be copied. To duplicate the entire container, pass a value of `kParentAtomIsContainer` for the *atom* parameter.

targetContainer

A pointer to an uninitialized atom container data structure. On return, this parameter points to an atom container that contains a copy of the atom.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The caller is responsible for disposing of the new atom container by calling `QTDisposeAtomContainer` (II–1164).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Copying Existing Atoms” (V–2767)

Related Java Methods

`quicktime.std.movies.AtomContainer.copyAtom()`

QTCopyAtomDataToHandle

Copies the specified leaf atom’s data to a handle.

```
OSErr QTCopyAtomDataToHandle (
    QTAtomContainer    container,
    QTAtom             atom,
    Handle              targetHandle );
```

container

The atom container that contains the leaf atom.

atom

The leaf atom whose data should be copied.

targetHandle

A handle. On return, the handle contains the atom's data. The handle must not be locked. This function resizes the handle, if necessary.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

You call this function, passing an initialized handle, to retrieve a copy of a leaf atom's data.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Retrieving Atoms and Atom Data” (V-2768)

Related Java Methods

`quicktime.std.movies.AtomContainer.copyAtomDataToHandle()`

Q

QTCopyAtomDataToPtr

Copies the specified leaf atom's data to a buffer.

```
OSErr QTCopyAtomDataToPtr (
    QTAtomContainer    container,
    QTAtom             atom,
    Boolean             sizeOrLessOK,
    long               size,
    void                *targetPtr,
    long               *actualSize );
```

QTCountChildrenOfType

`container`

The atom container that contains the leaf atom.

`atom`

The leaf atom whose data should be copied.

`sizeOrLessOK`

Specifies whether the function may copy fewer bytes than the number of bytes specified by the `size` parameter. The buffer may be larger than the amount of atom data if you set the value of this parameter to `TRUE`. You can determine the size of an atom's data by calling `QTGetAtomDataPtr` (II–1171).

`size`

The length, in bytes, of the buffer pointed to by the `targetPtr` parameter.

`targetPtr`

A pointer to a buffer. On return, the buffer contains the atom data.

`actualSize`

A pointer to a long integer which, on return, contains the number of bytes copied to the buffer.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You call this function, passing a data buffer, to retrieve a copy of a leaf atom's data. The buffer must be large enough to contain the atom's data.

Special Considerations

This function may move memory.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Retrieving Atoms and Atom Data” (V–2768)

Related Java Methods

`quicktime.std.movies.AtomContainer.copyAtomDataToPtr()`

QTCountChildrenOfType

Returns the number of atoms of a given type in the child list of the specified parent atom.

```
short QTCountChildrenOfType (
    QTAtomContainer    container,
    QTAtom             parentAtom,
    QTAtomType         childType );
```

container

The atom container that contains the parent atom.

parentAtom

The parent atom for this operation.

childType

The atom type for this operation. To retrieve the total number of atoms in the child list, set this parameter to 0.

function result The number of atoms of a given type in the child list of the specified parent atom.

Discussion

You can call this function to determine the number of atoms of a specified type in a parent atom's child list. If the total number of atoms in the parent atom's child list is 0, the parent atom is a leaf atom.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Retrieving Atoms and Atom Data" (V-2768)

Related Java Methods

`quicktime.std.movies.AtomContainer.countChildrenOfType()`

Q

QTCreateStandardParameterDialog

Creates a dialog box that allows the user to choose an effect from the list of effects passed to the function.

```
OSErr QTCreateStandardParameterDialog (
    QTAtomContainer    effectList,
    QTAtomContainer    parameters,
    QTParameterDialogOptions    dialogOptions,
    QTParameterDialog    *createdDialog );
```

QTCreateStandardParameterDialog

`effectList`

A list of the effects that the user can choose from. In most cases you should call `QTGetEffectsList` (II–1174) to generate this list. If you pass `NIL` in this parameter, the function calls `QTGetEffectsList` to retrieve the list of all currently installed effects; this list is then presented to the user.

`parameters`

An effect description containing the default parameter values for the effect. If the effect named in the parameter description is in `effectList`, that effect is displayed when the dialog is first shown and its parameter values are set from the parameter description. Pass in an empty atom container to have the dialog box display the first effect in the list, set to its default parameters. On return, this atom container holds an effect description for the effect selected by the user, including the parameter settings. This effect description can then be added to the media of an effect track. You will need to add source atoms to this container for effects that require sources.

`dialogOptions`

Options (see below) that control the behavior of the dialog.

`createdDialog`

Returns a reference to the dialog box that is created by this function. You should pass this value only to `QTIsStandardParameterDialogEvent` (II–1186) and `QTDismissStandardParameterDialog` (II–1163).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

dialogOptions Constants

`pdOptionsCollectOneValue`

Only one value can be entered for parameters that are optionally **tweenable**. This means that optionally-tweened parameters will not be tweened.

`pdOptionsAllowOptionalInterpolations`

The user interface for optionally-tweened parameters will be constructed to take **tweened** values. Setting this flag selects the “advanced” user interface mode. If the user interface supports this mode of operation, then a tween value can be entered for this parameter when the user interface is in the mode. An example of an advanced mode is the standard parameters dialog box; if you hold down the option key while selecting an effect, any parameters that have the `kAtomInterpolateIsOptional` flag set will allow a tween value to be entered.

Discussion

This function creates and displays a standard parameter dialog box that allows the user to choose an effect from the list in the `effectList` parameter. The dialog box also allows the user to choose values for the parameters of the effect, to **preview** the effects as they choose and customize them, and to get more information about each effect. Your application must call the Mac OS function `WaitNextEvent` and `QTIsStandardParameterDialogEvent` (II–1186) to allow the user to interact with the dialog box that is shown. Note that the dialog box will remain hidden until the first event is processed by `QTIsStandardParameterDialogEvent`. At this point, the dialog box will be displayed. You can modify the default behavior of the dialog box that is created by calling `QTStandardParameterDialogDoAction` (II–1313).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Effects Functions” (V–2897)

Related Java Methods

`quicktime.std.movies.ParameterDialog.showParameterDialog()`

QTDismissStandardParameterDialog

Closes a standard parameter dialog box that was created using `QTCreateStandardParameterDialog` (II–1161).

```
OSErr QTDismissStandardParameterDialog (
    QTParameterDialog    createdDialog );
```

`createdDialog`

The reference to the standard parameters dialog box that is returned by `QTCreateStandardParameterDialog` (II–1161).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

This function disposes of all memory associated with the dialog box.

Version Notes

Introduced in QuickTime 3 or earlier.

QTDisposeAtomContainer

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Effects Functions” (V–2897)

Related Java Methods

`quicktime.std.movies.ParameterDialog.showParameterDialog()`

QTDisposeAtomContainer

Disposes of an atom container.

```
OSErr QTDisposeAtomContainer (
    QTAtomContainer    atomData );
```

`atomData`

The atom container to be disposed of.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You can call this function to dispose of an atom container data structure that was created by `QTNewAtomContainer` (II–1203) or `QTCopyAtom` (II–1158).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Disposing of Atom Containers” (V–2766)

QTDisposeTween

Disposes of a **tween** component instance.

```
OSErr QTDisposeTween (
    QTTween    tween );
```

`tween`

The **tween** to be disposed of.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function

result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

QTDoTween

Runs a **tween** component.

```
OSErr QTDoTween (
    QTweener          tween,
    TimeValue         atTime,
    Handle             result,
    long               *resultSize,
    TweenerDataUPP    tweenDataProc,
    void               *tweenDataRefCon );
```

tween

The **tween** to be run.

atTime

A value that defines the time to run the **tween**.

result

A handle to the result of the **tweening** operation.

resultSize

A pointer to the size of the result.

tweenDataProc

A **Universal Procedure Pointer** that accesses a `TweenerDataProc` (III–2153) callback.

tweenDataRefCon

A pointer to a reference constant to be passed to your callback. Use this constant to point to a data structure containing any information your function needs.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

QTFindChildByID

Programming Info

C interface file: Movies.h

QTFindChildByID

Retrieves an atom by ID from the child list of the specified parent atom.

```
QTAtom QTFindChildByID (
    QTAtomContainer    container,
    QTAtom             parentAtom,
    QTAtomType         atomType,
    QTAtomID           id,
    short              *index );
```

container

The atom container that contains the parent atom.

parentAtom

The parent atom for this operation.

atomType

The type of the atom to be retrieved.

id

The ID of the atom to be retrieved.

index

A pointer to an uninitialized short integer. On return, if the atom specified by the *id* parameter was found, the integer contains the atom's index. If you don't want this function to return the atom's index, set the value of the *index* parameter to NIL.

function result The found atom.

Discussion

You call this function to search for and retrieve an atom by its type and ID from a parent atom's child list. The following code shows how you can use this function to insert a copy of container B's atoms as children of the 'abcd' atom in container A:

```
// QTFindChildByID coding example
QTAtom targetAtom;
targetAtom = QTFindChildByID (containerA, kParentAtomIsContainer, 'abcd',
    1000, NIL);
FailOSErr (QTInsertChildren (containerA, targetAtom, containerB));
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Retrieving Atoms and Atom Data” (V-2768)

Related Java Methods

`quicktime.std.movies.AtomContainer.findChildByID_Atom()`,
`quicktime.std.movies.AtomContainer.findChildByID_index()`

QTFindChildByIndex

Retrieves an atom by index from the child list of the specified parent atom.

```
QTAtom QTFindChildByIndex (
    QTAtomContainer    container,
    QTAtom             parentAtom,
    QTAtomType         atomType,
    short              index,
    QTAtomID           *id );
```

container

The atom container that contains the parent atom.

parentAtom

The parent atom for this operation.

atomType

The type of the atom to be retrieved.

index

The index of the atom to be retrieved.

id

A pointer to an uninitialized `QTAtomID` data structure. On return, if the atom specified by *index* was found, the `QTAtomID` data structure contains the atom’s ID. If you don’t want this function to return the atom’s ID, set the value of the *id* parameter to `NIL`.

function result The found atom.

Discussion

You call this function to search for and retrieve an atom by its type and index within that type from a parent atom’s child list. The following code illustrates one way to use it:

QTGetAccessKeys

```
// QTFindChildByIndex coding example
if ((propertyAtom = QTFindChildByIndex (sprite, kParentAtomIsContainer,
    kSpritePropertyImageIndex, 1, NIL)) == 0)

    FailOSErr (QTInsertChild (sprite, kParentAtomIsContainer,
        kSpritePropertyImageIndex, 1, 1, sizeof(short), &imageIndex,
        NIL));
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Retrieving Atoms and Atom Data” (V–2768)

Related Java Methods

```
quicktime.std.movies.AtomContainer.findChildByIndex_Atom(),
quicktime.std.movies.AtomContainer.findChildByIndex_id()
```

QTGetAccessKeys

Returns all the application and system access keys of a specified access key type.

```
OSErr QTGetAccessKeys (
    Str255          accessKeyType,
    long            flags,
    QTAtomContainer *keys );
```

accessKeyType

The type of access keys to return.

flags

Unused; must be set to 0.

keys

A pointer to a QT atom container that contains atoms of type `kAccessKeyAtomType` at the top level. These atoms contain the keys. If there are no access keys of the specified type, the function returns an empty QT atom container.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

In the QT atom container, application keys, which are more likely to be the ones an application needs, appear before system keys. You can get the key values by using QT atom functions.

Special Considerations

When your application is done with the QT atom container, it must dispose of it by calling `QTDisposeAtomContainer` (II–1164).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Retrieving Access Keys” (V–2771)

QTGetAliasInfo

Retrieves information from an **alias** record without actually resolving the alias.

```
OSErr QTGetAliasInfo (
    AliasHandle      alias,
    AliasInfoType    index,
    char             *outBuf,
    long             bufLen,
    long             *outLen,
    unsigned long    flags );
```

alias

A handle to an `AliasRecord` (IV–2159) structure.

index

The kind of information to be retrieved. If a positive integer is passed, the function retrieves the name of the parent directory that many levels above the **alias**. If the integer is greater than the number of levels to the root directory, an empty string is returned. You can also pass a constant (see below).

outBuf

A pointer to a buffer to hold the returned C string. This function returns strings of potentially unlimited length. If you pass `NIL` here, or set `bufLen` too short, the function will return an error and write the actual length needed to `outLen`. So you can make one call to this function, passing `NIL`, then allocate the buffer and call it again with the correct length.

QTGetAliasInfo

bufLen

The length of the actual output buffer.

outLen

The required output buffer length, including any trailing null character.

flags

Reserved; set to 0.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

index Constants

asiZoneName

The name of the zone containing the target of the **alias**. This selector will always return an empty string on the Windows platform.

asiServerName

The name of the server containing the target. This will always return an empty string in the usual case of a drive-letter based path. But sometimes these **aliases** refer to paths such as \\www.mypage.com\\dropboxes\\Bob Jones\\sample.mov, instead of a drive-letter based path. In those cases, asiServerName will return www.mypage.com and asiVolumeName will return dropboxes.

asiVolumeName

The name of the volume containing the target.

asiAliasName

The name of the target.

asiParentName

The name of the parent directory of the target. If the target is a volume, return the volume name.

Discussion

This function is used to retrieve the name of the target, the names of the target’s parent directories, the name of the target’s volume, or its zone or server name.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTML.h

See Also

See GetAliasInfo in *Inside Macintosh: Files* (listed in the **bibliography**).

QTGetAtomDataPtr

Retrieves a pointer to the atom data for a specified leaf atom.

```
OSErr QTGetAtomDataPtr (
    QTAtomContainer    container,
    QTAtom             atom,
    long               *dataSize,
    Ptr                *atomData );
```

container

The atom container that contains the leaf atom.

atom

The leaf atom whose data should be retrieved.

dataSize

On return, contains a pointer to the length, in bytes, of the leaf atom's data.

atomData

On return, contains a pointer to the leaf atom's data.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

You call this function to retrieve a pointer to a leaf atom's data so that you can access the data directly.

Special Considerations

To ensure that the pointer returned in the `atomData` parameter will remain valid if memory is moved, you should call `QTLockContainer` (II-1187) before you call this function. If you call `QTLockContainer`, you should call `QTUnlockContainer` (II-1316) when you have finished using the `atomData` pointer. If you pass a locked atom container to a function that resizes atom containers, the function returns an error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Retrieving Atoms and Atom Data” (V-2768)

Related Java Methods

`quicktime.std.movies.AtomContainer.getAtomData()`

QTGetAtomParent

QTGetAtomParent

Gets the parent of a QT atom.

```
QTAtom QTGetAtomParent (
    QTAtomContainer    container,
    QTAtom             childAtom );
```

container

A QT atom container.

childAtom

A QT child atom in the container.

function result On return, the parent of the child atom.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

QTGetAtomTypeAndID

Retrieves an atom's type and ID.

```
OSErr QTGetAtomTypeAndID (
    QTAtomContainer    container,
    QTAtom             atom,
    QTAtomType         *atomType,
    QTAtomID           *id );
```

container

The atom container that contains the atom.

atom

The atom whose type and ID should be retrieved.

atomType

A pointer to an atom type. On return, this parameter points to the type of the specified atom. You can pass `NIL` for this parameter if you don't need this information.

id

A pointer to an atom ID. On return, this parameter points to the ID of the specified atom. You can pass NIL for this parameter if you don't need this information.

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Retrieving Atoms and Atom Data” (V-2768)

Related Java Methods

quicktime.std.movies.AtomContainer.getAtomType(),
quicktime.std.movies.AtomContainer.getAtomID()

QTGetDataRefMaxFileOffset

Undocumented

```
OSErr QTGetDataRefMaxFileOffset (
    Movie      movieH,
    OSType     dataRefType,
    Handle     dataRef,
    long       *offset );
```

movieH

Undocumented

dataRefType

The type of data reference; see “Data References” (IV-2661). If the data reference is an **alias**, you must set this parameter to rAliasType. See *Inside Macintosh: Files* (listed in the **bibliography**) for more information about aliases and the Alias Manager.

dataRef

A handle to a data reference. The type of information stored in the handle depends upon the data reference type specified by the dataRefType parameter.

offset

Undocumented

QTGetDDObject

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

QTGetDDObject

Returns the Windows DirectDraw object currently in use by QuickTime.

```
OSErr QTGetDDObject (
    void      **lpDDObject );
```

`lpDDObject`

A handle to a DirectDraw object.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is useful for developers who want to call DirectDraw methods directly.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QTML.h`

See Also

For the corresponding set function, see `QTSetDDObject` (II–1225).

QTGetEffectsList

Returns a QT atom container holding a list of the currently installed effects components.

```
OSErr QTGetEffectsList (
    QTAAtomContainer    *returnedList,
    long                minSources,
    long                maxSources,
    QTEffectListOptions getOptions );
```

`returnedList`

If the function returns `noErr`, this parameter contains a newly created QT atom container holding a list of their currently installed effects. Any data stored in the parameter on entry is overwritten by the list of effects. It is the responsibility of the calling application to dispose of the storage by calling `QTDisposeAtomContainer` (II–1164) once the list is no longer required.

`minSources`

The minimum number of sources that an effect must have to be added to the list. Pass `–1` as this parameter to specify no minimum.

`maxSources`

The maximum number of sources that an effect can have to be added to the list. Pass `–1` as this parameter to specify no maximum. The `minSources` and `maxSources` parameters allow you to restrict which effects are returned in the list, by specifying the minimum and maximum number of sources that qualifying effects can have.

`getOptions`

Options (see below) that control which effects are added to the list. If you pass `0`, the function includes every effect, except the “none” effect and any prohibited by the values of `minSources` and `maxSources`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

getOptions Constants

`e1OptionsIncludeNoneInList`

Include the “none” effect in the returned list. The “none” effect does nothing.

Discussion

The returned list contains two atoms for each effect component. The first atom, of type `kEffectNameAtom`, contains the name of the effect. The second atom, of type `kEffectTypeAtom`, contains the type of the effect, which is the sub-type of the effect component. This list is sorted alphabetically on the names of the effects. You can constrain the list to certain types of effects, such as those that take two sources. Use this function to obtain a list of effects that you can pass to `QTCreateStandardParameterDialog` (II–1161).

Special Considerations

This function can take a fairly long time to execute, as it searches the system for installed effects components. You will normally want to call this function once when your application starts, or after a pair of suspend and resume events.

QTGetEffectSpeed

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Effects Functions” (V–2897)

QTGetEffectSpeed

Returns the speed of the effect, expressed in frames per second.

```
OSErr QTGetEffectSpeed (
    QTAtomContainer    parameters,
    Fixed              *pFPS );
```

parameters

Contains parameter values for the effect.

pFPS

The speed of the effect is returned in this parameter, expressed in frames per second. Effects can also return the pre-defined constant `effectIsRealtime` (see below) as their speed.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

pFPS Constant

`effectIsRealtime`

The effect will execute as quickly as frames can be sent to it for rendering.

Discussion

The value returned should not be treated as an absolute measurement of effect performance. In particular, most effects only return one value, regardless of parameter settings and hardware. This value is an estimate of execution speed on a reference hardware platform. Actual performance will vary depending on hardware, configuration and parameter options.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Effects Functions” (V–2897)

QTGetFileNameExtension

Gets the extension to a file name.

```
OSErr QTGetFileNameExtension (
    ConstStrFileNameParam    fileName,
    OSType                   fileType,
    OSType                   *extension );
```

fileName

A file name string.

fileType

A file type; see “File Types and Creators” (IV–2668).

extension

A pointer to the file name extension string.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

QTGetMIMETypeInfo

Retrieves information about a particular MIME type.

```
OSErr QTGetMIMETypeInfo (
    const char    *mimeStringStart,
    short         mimeStringLength,
    OSType        infoSelector,
    void          *infoDataPtr,
    long          *infoDataSize );
```

mimeStringStart

A pointer to the first character of a string holding the MIME type.

mimeStringLength

The number of characters in the MIME type string. Pascal, C, and nondelimited string buffers can be passed equally well.

infoSelector

A constant (see below) that indicates the type of information being requested.

QTGetNextChildType

infoDataPtr

A pointer to a value to be updated. For current selectors this value is Boolean.

infoDataSize

On input, a pointer to the size of the data being expected; on output, a pointer to the size of the data being retrieved. In all current cases these will be the same size.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

infoSelector Constants

kQTGetMIMETYPEInfoIsQuickTimeMovieType

Corresponds to a MIME type for a QuickTime movie. The current check is against video/quicktime and application/x-quicktimeplayer but this may be extended in the future.

kQTGetMIMETYPEInfoIsUnhelpfulType

Useful in trying to determine an importer, this returns false for application/octet-stream, a MIME type that often indicates a poorly configured server. This allows the MIME check to be bypassed for obviously bogus MIME type information.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `Movies.h`

QTGetNextChildType

Returns the next atom type in the child list of the specified parent atom.

```
QTAtomType QTGetNextChildType (
    QTAtomContainer    container,
    QTAtom             parentAtom,
    QTAtomType         currentChildType );
```

container

The atom container that contains the parent atom.

parentAtom

The parent atom for this operation.

`currentChildType`

The last atom type retrieved by this function.

function result The next atom type in the child list of the atom specified by `parentAtom`.

Discussion

You can call this function to iterate through the atom types in a parent atom's child list. To retrieve the first atom type, you should set the value of the `currentChildType` parameter to 0. To retrieve subsequent atom types, you should set the value of the `currentChildType` parameter to the atom type retrieved by the previous call to this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Retrieving Atoms and Atom Data” (V–2768)

Related Java Methods

`quicktime.std.movies.AtomContainer.getNextChildType()`

QTGetPixelSize

Returns the bits per pixel for a given pixel format.

```
short QTGetPixelSize (
    OSType    PixelFormat );
```

`PixelFormat`

A constant that identifies the pixel format; see “Pixel Formats” (IV–2686). This function returns meaningful information only for non-planar formats.

function result The bits per pixel. Returns 0 if the format is unknown.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Related Java Methods

`quicktime.qd.QDGraphics.getPixelSize()`

QTGetPixMapHandleGammaLevel

See Also

See the `ICMSetPixelFormatInfo` (II–678) function and the `ICMPixelFormatInfo` (IV–2282) structure.

QTGetPixMapHandleGammaLevel

Retrieves the current `PixMap` extension’s gamma level setting.

Fixed `QTGetPixMapHandleGammaLevel (PixMapHandle pm);`

`pm`

A handle to a `PixMap` (IV–2332) structure that has a `PixMapExtension` (IV–2335) structure.

function result On return, the gamma level previously set (or the default level) for the pixel map referenced by the `pm` parameter.

Discussion

A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding set function, see `QTSetPixMapHandleGammaLevel` (II–1227).

QTGetPixMapHandleRequestedGammaLevel

Retrieves the current `PixMap` extension’s requested gamma level setting.

Fixed `QTGetPixMapHandleRequestedGammaLevel (PixMapHandle pm);`

`pm`

A handle to a `PixMap` (IV–2332) structure that has a `PixMapExtension` (IV–2335) structure.

function result On return, the requested gamma level previously set (or the default level) for the pixel map referenced by the `pm` parameter.

Discussion

A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure. The requested gamma level is used to control what gamma conversion is attempted during decompression. The requested gamma level may differ from the actual gamma level depending on the compressed data and the capabilities of the codecs involved.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding set function, see `QTSetPixMapHandleRequestedGammaLevel` (II-1227).

QTGetPixMapHandleRowBytes

Gets the `rowBytes` value for a pixel map accessed by a handle.

```
long QTGetPixMapHandleRowBytes (
    PixMapHandle  pm );
```

pm

A handle to a `PixMap` (IV-2332) structure.

function result The `rowBytes` value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding set function, see `QTSetPixMapHandleRowBytes` (II-1228).

QTGetPixMapPtrGammaLevel

Retrieves the current `PixMap` extension's gamma level setting.

```
Fixed QTGetPixMapPtrGammaLevel (PixMapPtr pm);
```

QTGetPixMapPtrRequestedGammaLevel

pm

A pointer to a `PixMap` (IV–2332) structure that has a `PixMapExtension` (IV–2335) structure.

function result On return, the gamma level previously set (or the default level) for the pixel map pointed to by the *pm* parameter.

Discussion

A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding set function, see `QTSetPixMapPtrGammaLevel` (II–1229).

QTGetPixMapPtrRequestedGammaLevel

Retrieves the current `PixMap` extension’s gamma level setting.

Fixed `QTGetPixMapPtrRequestedGammaLevel (PixMapPtr pm);`

pm

A pointer to a `PixMap` (IV–2332) structure that has a `PixMapExtension` (IV–2335) structure.

function result On return, the requested gamma level previously set (or the default level) for the pixel map pointed to by the *pm* parameter.

Discussion

A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure. The requested gamma level is used to control what gamma conversion is attempted during decompression. The requested gamma level may differ from the actual gamma level depending on the compressed data and the capabilities of the codecs involved.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding set function, see QTSetPixMapPtrRequestedGammaLevel (II–1230).

QTGetPixMapPtrRowBytes

Gets the rowBytes value for a pixel map accessed by a pointer.

```
long QTGetPixMapPtrRowBytes (
    PixMapPtr    pm );
```

pm

A pointer to a PixMap (IV–2332) structure.

function result The rowBytes value.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

See Also

For the corresponding set function, see QTSetPixMapPtrRowBytes (II–1231).

QTInsertChild

Creates a new child atom of the specified parent atom.

```
OSErr QTInsertChild (
    QAtomContainer    container,
    QAtom             parentAtom,
    QAtomType         atomType,
    QAtomID           id,
    short             index,
    long              dataSize,
    void              *data,
    QAtom             *newAtom );
```

container

The atom container that contains the parent atom. The atom container must not be locked.

QTInsertChild

parentAtom

The parent atom within the atom container.

atomType

The type of the new atom to be inserted.

id

The ID of the new atom to be inserted. This ID must be unique among atoms of the same type for the specified parent. If you set this parameter to 0, the function assigns a unique ID to the atom.

index

The index of the new atom among atoms with the same parent. To insert the first atom for the specified parent, you should set this parameter to 1. To insert an atom as the last atom in the child list, you should set this parameter to 0. Index values greater than the index of the last atom in the child list plus 1 are invalid.

dataSize

The size of the data for the new atom. If the new atom is to be a parent atom or if you want to add the atom's data later, you should pass 0 for this parameter. To create the new atom as a leaf atom that contains data, you should specify the data using the data parameter and its size using the dataSize parameter.

data

A pointer to a buffer containing the data for the new atom. If you set the value of the dataSize parameter to 0, you should pass `NIL` for this parameter.

newAtom

A pointer to data of type `QTAtom`. On return, this parameter points to the newly created atom. You can pass `NIL` for this parameter if you don't need a reference to the newly created atom.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See "Error Codes" (IV-2663).

Discussion

You call this function to create a new child atom. The new child atom has the specified atom type and atom ID, and is inserted into its parent atom's child list at the specified index. Any existing atoms at the same index or greater are moved toward the end of the child list.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating New Atoms” (V–2767)

Related Java Methods

`quicktime.std.movies.AtomContainer.insertChild()`

QTInsertChildren

Inserts a container of atoms as children of the specified parent atom.

```
OSErr QTInsertChildren (
    QTAtomContainer    container,
    QTAtom             parentAtom,
    QTAtomContainer    childrenContainer );
```

`container`

The atom container that contains the parent atom. The atom container must not be locked.

`parentAtom`

The parent atom within the atom container.

`childrenContainer`

The atom container that contains the child atoms to be inserted.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You call this function to insert a container of atoms as children of a parent atom in another atom container. Each child atom is inserted as the last atom of its type and is assigned a corresponding index. The ID of a child atom to be inserted must not duplicate that of an existing child atom of the same type. The following code shows how you can use this function to create a container, insert an atom, and insert another container as a child of the atom:

```
// QTInsertChildren coding example
FailOSErr (QTInsertChild (outerContainer, kParentAtomIsContainer,
    kSpriteAtomType, spriteID, 0, 0, NIL, &newParentAtom));
FailOSErr (QTInsertChildren (outerContainer, newParentAtom,
    innerContainer));
```

QTIsStandardParameterDialogEvent

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Copying Existing Atoms” (V–2767)

Related Java Methods

`quicktime.std.movies.AtomContainer.insertChildren()`

QTIsStandardParameterDialogEvent

Determines if a Macintosh event is processed by a standard parameter dialog box created by `QTCreatStandardParameterDialog` (II–1161).

```
OSErr QTIsStandardParameterDialogEvent (  
    EventRecord      *pEvent,  
    QTParameterDialog  createdDialog );
```

`pEvent`

The Macintosh event.

`createdDialog`

The reference to the standard parameters dialog box that is returned by `QTCreatStandardParameterDialog` (II–1161).

function result See below.

Function Result Constants

`noErr`

The event was related to the standard parameters dialog box and was completely processed. Your application should not process this event.

`featureUnsupported`

The event was not related to the standard parameters dialog box. Your application should process the event in the normal way.

`codecParameterDialogConfirm`

The user clicked the OK button in the standard parameters dialog box. Your application should call `QTDismissStandardParameterDialog` (II–1163) to close the dialog box. The values chosen by the user are put into the atom container you passed in the `parameters` parameter when you called `QTCreatStandardParameterDialog` (II–1161). This atom container now holds an effect description that is ready to insert into an effects track. It may require one or two 'src ' atoms to be complete.

userCanceledErr

The event was handled by the standard parameter dialog box, and the user clicked the Cancel button of the dialog box. You should call QTDismissStandardParameterDialog (II–1163) to close the dialog box.

Discussion

After you create a standard parameter dialog box, pass every Macintosh event through this function to determine if your application should handle the event. Once the dialog box has been confirmed or cancelled by the user, you should no longer call this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “High-Level Effects Functions” (V–2897)

Related Java Methods

quicktime.std.movies.ParameterDialog.showParameterDialog()

QTLockContainer

Locks an atom container in memory.

```
OSErr QTLockContainer (
    QTAtomContainer  container );
```

container

The atom container to be locked.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You should call this function to lock an atom container before calling QTGetAtomDataPtr (II–1171) to directly access a leaf atom’s data. When you have finished accessing a leaf atom’s data, you should call QTUnlockContainer (II–1316). You may make nested pairs of calls to QTLockContainer and QTUnlockContainer; you don’t need to check the current state of the container first.

Version Notes

Introduced in QuickTime 3 or earlier.

QTMIDIGetMIDIPorts

Programming Info

C interface file: `Movies.h`

Programming summary: “Retrieving Atoms and Atom Data” (V–2768)

QTMIDIGetMIDIPorts

Returns two lists of MIDI ports supported by the specified MIDI component: a list of ports that can receive MIDI input and a list of ports that can send MIDI output.

```
ComponentResult QTMIDIGetMIDIPorts (
    QTMIDIComponent          ci,
    QTMIDIPortListHandle      *inputPorts,
    QTMIDIPortListHandle      *outputPorts );
```

ci

A MIDI component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

inputPorts

A list of the MIDI ports supported by the component that can receive MIDI input.

outputPorts

A list of the MIDI ports supported by the component that can send MIDI output.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The caller of this function must dispose of the `inputPorts` and `outputPorts` handles.

Special Considerations

`NAGetMIDIPorts` (II–1025) is the correct call for applications to make. They should not call this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “MIDI Component Functions” (V–2924)

QTMIDISendMIDI

Sends MIDI data to a MIDI port.

```
ComponentResult QTMIDISendMIDI (
    QTMIDIComponent    ci,
    long                portIndex,
    MusicMIDIPacket     *mp );
```

ci

A MIDI component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

portIndex

The index of the MIDI port to use for this operation.

mp

A pointer to the MIDI data packet to send.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function can be called at interrupt time. However, the same interrupt level is used whenever MIDI data is sent by the specified MIDI component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “MIDI Component Functions” (V–2924)

QTMIDIUseReceivePort

Allocates a MIDI port for input or to release the port.

```
ComponentResult QTMIDIUseReceivePort (
    QTMIDIComponent    ci,
    long                portIndex,
    MusicMIDIReadHookUPP readHook,
    long                refCon );
```

ci

A MIDI component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

QTMIDIUseSendPort

portIndex

The index of the MIDI port to use for this operation.

readHook

A pointer to a `MusicMIDIReadHookProc` (III–2109) callback that receives incoming MIDI data packets, or `NIL` to release the port.

refCon

A reference constant passed to the function specified by the `readHook` parameter. Use this parameter to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The MIDI component delivers only MIDI data packets that contain a single status byte.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “MIDI Component Functions” (V–2924)

QTMIDIUseSendPort

Allocates a MIDI port for output or to release the port.

```
ComponentResult QTMIDIUseSendPort (
    QTMIDIComponent    ci,
    long                portIndex,
    long                inUse );
```

ci

A MIDI component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

portIndex

The index of the MIDI port for this operation.

inUse

Specifies whether to allocate the MIDI port for output (if the value is 1) or to release the port (if the value is 0).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “MIDI Component Functions” (V-2924)

QTMLAcquireWindowList

Acquires exclusive access to the global list of Macintosh windows, so that the list will not change until you call `QTMLReleaseWindowList` (II-1196).

```
void QTMLAcquireWindowList ( void );
```

Discussion

If you want to call the Windows `LMGetWindowList` or `FrontWindow` functions and then proceed to walk down the next pointers, you need to protect yourself against another thread modifying the list while you walk. Call this function before and `QTMLReleaseWindowList` (II-1196) after.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QTML.h`

QTMLCreateMutex

Creates a synchronization object to facilitate mutually exclusive access to a Windows data structure.

```
QTMLMutex QTMLCreateMutex ( void );
```

Discussion

This function creates a mutex object for guarded access to data structures and routines that require mutually exclusive access. In a multithreaded preemptive environment, such as Windows NT, you can use the various mutex utility functions such as `QTMLGrabMutex` (II-1195) to protect a shared **resource** from simultaneous access by multiple threads or processes. Mutex objects are used throughout QTML to provide such protection.

Version Notes

Introduced in QuickTime 3 or earlier.

QTMLCreateSyncVar

Programming Info

C interface file: QTML.h

QTMLCreateSyncVar

Creates a synchronization variable, used to provide guarded access to **resources** shared across threads and processes.

```
QTMLSyncVarPtr QTMLCreateSyncVar ( void );
```

function result A pointer to a Windows synchronization variable.

Discussion

You call this function to create a synchronization variable that allows for mutually-exclusive access to **resources**. The synchronization variable routines use atomic tests to ensure that the portions of the routines that perform the testing cannot be interrupted during the test.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

QTMLDestroyMutex

Deallocates a synchronization object created by QTMLCreateMutex (II-1191).

```
void QTMLDestroyMutex (
    QTMLMutex    mu );
```

mu

A mutex object.

Discussion

Call this function to deallocate the mutex object created by QTMLCreateMutex (II-1191).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

QTMLDestroySyncVar

Releases ownership of a synchronization variable.

```
void QTMLDestroySyncVar (
    QTMLSyncVarPtr    p );
```

p

A pointer to a synchronization variable.

Discussion

Call this function to deallocate the QTMLSyncVar object created by QTMLCreateSyncVar (II–1192).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

QTMLGetCanonicalPathName

Takes a Windows native file path and returns the one canonical path to that file.

```
OSErr QTMLGetCanonicalPathName (
    char          *inName,
    char          *outName,
    unsigned long  outLen );
```

inName

The input path.

outName

Specifies where the routine puts the canonical path.

outLen

The length of the outName buffer, so that the routine knows not to write past the end of your buffer.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Some of the tasks performed by this routine include removing all double-dots from the path, converting all short (8.3) names back to their long names, and restoring the correct capitalization. This routine also works for universal naming convention (UNC) paths. These paths are of the form:

QTMLGetVolumeRootPath

```
c:\Program Files\Some Product\test.mov
c:\PROGRA-1\SOMEPR-1\test.mov
c:\Some other folder\..\program FILES\another program\..\somepr-1\TeSt.MoV
C:\proGra-1\Some product\..\SOMEPR-1\...\program files\a_product\test.mov
C:\PROGRA-1\SOMEPR-1\TEST.MOV
c:\Program Files\A_Product\test.mov
\\my_server\shared_folder\a_folder\test.mov
```

Special Considerations

There is a one-to-one mapping between canonical pathnames and files. In other words, you can determine if two paths point to the same file by canonicalizing both paths, and then doing a string compare.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

QTMLGetVolumeRootPath

Takes a Windows path and returns the portion of it that points to the volume root.

```
OSErr QTMLGetVolumeRootPath (
    char          *fullPath,
    char          *volumeRootPath,
    unsigned long  volumeRootLen );
```

fullPath

The path being passed in.

volumeRootPath

Specifies where this routine writes the volume root path.

volumeRootLen

The length of the *volumeRootPath* buffer, so the routine knows not to write past the end of your buffer.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Discussion

This routine is useful when you need to call Windows routines, such as *GetVolumeInformation*, which take a volume root path as an argument.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

QTMLGetWindowWndProc

Returns the WndProc previously specified in QTMLSetWindowWndProc (II–1198).

```
void * QTMLGetWindowWndProc (
    WindowPtr    theWindow );
```

theWindow

The Macintosh window to hook. This must have been created via the Windows calls NewCWindow or NewWindow, or as a result of calling CreatePortAssociation (I–148) on a native HWND.

function result Returns NIL if no application-defined WndProc is set.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTML.h

See Also

For the corresponding set function, see QTMLSetWindowWndProc (II–1198).

QTMLGrabMutex

Confers ownership of a mutex created by QTMLCreateMutex (II–1191).

```
void QTMLGrabMutex (
    QTMLMutex    mu );
```

mu

A mutex object.

Discussion

Call this function when you require exclusive ownership of the **resource** guarded by a mutex. This function will return when you have gained this ownership. In the case where another thread or process holds the mutex, this function waits until that process or thread relinquishes control. If you need to determine if you can grab the mutex, without actually grabbing it, call QTMLTryGrabMutex (II–1199).

QTMLRegisterInterruptSafeThread

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

QTMLRegisterInterruptSafeThread

Registers a thread of execution that is allowed to make interrupt-safe calls.

```
long QTMLRegisterInterruptSafeThread (
    unsigned long    threadID,
    void             *threadInfo );
```

threadID

Thread ID of the calling thread. This value is obtained by calling the Win32 GetCurrentThreadId function.

threadInfo

Thread information. This value is obtained by calling the Win32 GetCurrentThread function.

function result Returns noErr if successful.

Discussion

The QTML function dispatcher includes a mechanism that prevents not only the QuickTime Toolbox from being reentered but also allows certain APIs to be callable at interrupt time. On the Macintosh, these calls require that the calling code not allocate, move, or purge memory. On Windows, threads that emulate interrupt handlers need to register with QTML by calling this function, so that API calls made from that thread are not blocked. You should make this call at the top of your thread main routine.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

QTMLReleaseWindowList

Allows Windows threads to modify the global list of Macintosh-style windows.

```
void QTMLReleaseWindowList ( void );
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

QTMLResetSyncVar

Resets the lock for a QTMLSyncVar object.

```
void QTMLResetSyncVar (
    QTMLSyncVarPtr    sync );
```

sync

A pointer to a synchronization variable.

Discussion

Call QTMLResetSyncVar to relinquish the lock obtained from a previous call to QTMLWaitAndSetSyncVar (II–1200).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

QTMLReturnMutex

Releases ownership of a QTMLMutex object.

```
void QTMLReturnMutex (
    QTMLMutex    mu );
```

mu

A mutex object.

Discussion

Call this function to balance a call to QTMLGrabMutex (II–1195) when you are ready to relinquish control of the mutex and corresponding shared **resource**. By making this call, you allow other processes or threads waiting for the release of this mutex to gain access.

Version Notes

Introduced in QuickTime 3 or earlier.



QTMLSetWindowWndProc

Programming Info

C interface file: QTML.h

QTMLSetWindowWndProc

Specifies an application-defined window procedure (WndProc) which is called by QTML after QTML processes the message for an HWND.

```
void QTMLSetWindowWndProc (  
    WindowPtr    theWindow,  
    void         *windowProc );
```

theWindow

A pointer to the Macintosh window to hook. This must have been created via the Windows calls NewCWindow or NewWindow, or as a result of calling CreatePortAssociation (I-148) on a native HWND.

windowProc

A Windows WndProc procedure. For a detailed description of the WndProc procedure, see the Win32 documentation.

Discussion

This routine is useful if you want to perform some special Windows processing of the native messages that Windows sends to your window pointer.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTML.h

See Also

For the corresponding get function, see QTMLGetWindowWndProc (II-1195).

QTMLTestAndSetSyncVar

Performs a one-shot atomic test and set operation of a QTMLSyncVar object.

```
long QTMLTestAndSetSyncVar (  
    QTMLSyncVarPtr    sync );
```

sync

A pointer to a synchronization variable.

function result Returns 0 if you have acquired the synchronization lock.

Discussion

Call this function to perform a single test and set operation on the QTMLOriginSyncVar object.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTMLOrigin.h

QTMLOriginTryGrabMutex

Determines if you would be able to get immediate ownership of a mutex created by QTMLOriginCreateMutex (II-1191).

```
Boolean QTMLOriginTryGrabMutex (  
    QTMLOriginMutex    mu );
```

mu

A mutex object.

function result Returns TRUE if you are able to immediately grab the mutex, via the QTMLOriginGrabMutex (II-1195) call, without having to wait.

Discussion

Call this function when you need to preflight a QTMLOriginGrabMutex (II-1195) call.

Special Considerations

Under normal circumstances you should not need to make this call.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTMLOrigin.h

QTMLOriginUnregisterInterruptSafeThread

Unregisters a thread of execution.

```
long QTMLOriginUnregisterInterruptSafeThread (  
    unsigned long    threadID );
```

QTMLWaitAndSetSyncVar

threadID

Thread ID of the calling thread. This value is obtained by calling the Win32 `GetCurrentThreadId` function.

function result Returns `noErr` if successful.

Discussion

Use this function to unregister an interrupt-safe thread previously registered by `QTMLRegisterInterruptSafeThread` (II–1196). You should make this call at the bottom of your thread main routine, just before the exit.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QTML.h`

QTMLWaitAndSetSyncVar

Acquires the lock for a `QTMLSyncVar` object,

```
void QTMLWaitAndSetSyncVar (  
    QTMLSyncVarPtr    sync );
```

sync

A pointer to a synchronization variable.

Discussion

Call this function to acquire the lock corresponding to a `QTMLSyncVar` object. This function will wait, yielding CPU time to other threads, until the lock is acquired.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QTML.h`

QTMLYieldCPU

Yields time to other threads while your code is in a tight loop.

```
void QTMLYieldCPU ( void );
```

Discussion

Use this function from within tight loops to yield time to other threads. Using this function is similar to calling `SystemTask` from within a Macintosh event loop.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QTML.h`

QTMLYieldCPUTime

Yields time to other threads and specifies the sleep time while in a tight loop.

```
void QTMLYieldCPUTime (
    long          milliseconds,
    unsigned long  flags );
```

`milliseconds`

Number of milliseconds to sleep before returning to the caller.

`flags`

A flag (see below) that specifies an option for this function.

flags Constant

`kQTMLHandlePortEvents`

If this flag is set, QTML will call the Win32 functions `PeekMessage`, `TranslateMessage`, and `DispatchMessage` to process Win32 messages while in tight spin loops.

Discussion

Use this function from within tight loops to yield time to other threads.

Special Considerations

This function differs from `QTMLYieldCPU` (II-1200) in that you can specify the time to sleep as well as optionally have QTML process Win32 messages while waiting for the yield time to expire.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QTML.h`

QTMovieNeedsTimeTable

QTMovieNeedsTimeTable

Returns whether a movie is being progressively downloaded.

```
OSErr QTMovieNeedsTimeTable (
    Movie      theMovie,
    Boolean     *needsTimeTable );
```

theMovie

The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

needsTimeTable

If TRUE, the movie is being progressively downloaded. If an error occurs, this parameter is set to FALSE.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

A movie can be progressively downloaded when its data is received over a network connection or other slow data channel. Progressive downloads are not necessary when the data for the movie is on a local disk. The Movie Toolbox creates a time table for a movie when either this function or `GetMaxLoadedTimeInMovie` (I–423) is called for the movie, but the time table is used only by the toolbox and is not accessible to applications. The toolbox disposes of the time table when the download is complete.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “High-Level Download Control” (V–2772)

Related Java Methods

`quicktime.std.movies.Movie.needsTimeTable()`

QTNewAlias

Creates a Mac OS **alias** to a file.

```
OSErr QTNewAlias (
```



```
const FSSpec      *fss,
AliasHandle      *alias,
Boolean          minimal );
```

fss

A pointer to an FSSpec (IV–2262) structure that specifies a file.

alias

On return, a pointer to a handle to a new AliasRecord (IV–2159) structure that defines an **alias** to the file. If the function was unable to create an alias, the handle is set to NIL. This function does not create relative aliases. For further information about Mac OS file aliases, see Chapter 4 of *Inside Macintosh: Files* (listed in the **bibliography**).

minimal

If you pass TRUE, the function writes in the AliasRecord structure only the target name, parent directory ID, volume name and creation date, and volume mounting information. If you pass FALSE, it fills out the structure fully.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Related Java Methods

quicktime.io.QTFile.newAlias()

Q

QTNewAtomContainer

Creates a new atom container.

```
OSErr QTNewAtomContainer (
    QTAtomContainer *atomData );
```

atomData

A pointer to an unallocated atom container data structure. On return, this parameter points to an allocated atom container.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function

QTNewGWorld

result. See “Error Codes” (IV–2663).

Discussion

This function creates a new, empty atom container structure. Once you have created an atom container, you can manipulate it using the atom container functions. The following example illustrates using this function to create a new QT atom container and add an atom:

```
// QTNewAtomContainer coding example
QTAtom firstAtom;
QTAtomContainer container;
OSErr err
err = QTNewAtomContainer (&container);
if (!err)
    err = QTInsertChild (container, kParentAtomIsContainer, 'abcd',
        1000, 1, 0, NIL, &firstAtom);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Disposing of Atom Containers” (V–2766)

Related Java Methods

```
quicktime.std.movies.AtomContainer.AtomContainer(),
quicktime.std.music.AtomicInstrument.AtomicInstrument(),
quicktime.std.movies.EffectsList.EffectsList()
```

QTNewGWorld

Creates an offscreen graphics world that may have a non-Macintosh pixel format.

```
OSErr QTNewGWorld (
    GWorldPtr      *offscreenGWorld,
    OSType         PixelFormat,
    const Rect      *boundsRect,
    CTabHandle      cTable,
    GDHandle        aGDevice,
    GWorldFlags     flags );
```

offscreenGWorld

On return, a pointer to the offscreen graphics world created by this routine.

PixelFormat

The new graphics world's pixel format; see "Pixel Formats" (IV-2686). This function won't work with planar pixel formats; use `QTNewGWorldFromPtr` (II-1206) instead. See the `ICMPixelFormatInfo` (IV-2282) structure for a discussion of planar and chunky formats.

boundsRect

A pointer to the boundary rectangle and port rectangle for the offscreen pixel map. This becomes the boundary rectangle for the `GDevice` structure, if this function creates one. If you specify 0 in the `PixelFormat` parameter, the function interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. It then uses the pixel format, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle. Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

cTable

A handle to a `ColorTable` (IV-2210) structure. If you pass `NIL` in this parameter, the function uses the default color table for the pixel format that you specify in the `PixelFormat` parameter. If you set the `PixelFormat` parameter to 0, the function ignores the `cTable` parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you use this function on a computer that supports only basic QuickDraw, you may specify only `NIL` in this parameter.

aGDevice

A handle to a `GDevice` (IV-2264) structure that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case the function attaches this structure to the new offscreen graphics world. If you set the `PixelFormat` parameter to 0, or if you do not set the `noNewDevice` flag, the function ignores this parameter, so you should set it to `NIL`. If you set the `PixelFormat` parameter to 0, the function uses the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NIL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` structures for offscreen graphics worlds.

flags

Constants (see below) that identify options available to your application. You can set a combination of these flags. If you don't wish to use any of them, pass



QTNewGWorldFromPtr

0 in this parameter. In this case the default behavior is to create an offscreen graphics world where the base address for the offscreen pixel image is unpurgeable, the graphics world uses an existing GDevice structure (if you pass 0 in the depth parameter) or creates a new GDevice structure, it uses memory in your application heap, and it allows graphics accelerators to cache the offscreen pixel image.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

flags Constants

pixPurge

Make base address for the offscreen pixel image purgeable.

noNewDevice

Do not create an offscreen GDevice structure.

useTempMem

Create a base address for the offscreen pixel image in temporary memory.

keepLocal

Keep the offscreen pixel image in main memory where it cannot be cached to a graphics accelerator card.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Related Java Methods

quicktime.qd.QDGraphics.QDGraphics()

QTNewGWorldFromPtr

Wraps a graphics world and pixel map structure around an existing block of memory containing an image.

```
OSErr QTNewGWorldFromPtr (
    GWorldPtr      *gw,
    OSType          pixelFormat,
    const Rect      *boundsRect,
    CTabHandle      cTable,
    GDHandle        aGDevice,
    GWorldFlags     flags,
    void            *baseAddr,
    long            rowBytes );
```

gw

On entry, a pointer that isn't going to change during the lifetime of the allocated graphics world. On return, this pointer references the offscreen graphics world created by this function.

pixelFormat

The new graphics world's pixel format; see "Pixel Formats" (IV-2686). This function won't work with planar pixel formats; see the `ICMPixelFormatInfo` (IV-2282) structure for a discussion of planar and chunky formats.

boundsRect

A pointer to the boundary rectangle and port rectangle for the offscreen pixel map. This becomes the boundary rectangle for the `GDevice` structure, if this function creates one. If you specify 0 in the `pixelFormat` parameter, the function interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. It then uses the pixel format, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle. Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

cTable

A handle to a `ColorTable` (IV-2210) structure. If you pass `NIL` in this parameter, the function uses the default color table for the pixel format that you specify in the `pixelFormat` parameter. If you set the `pixelFormat` parameter to 0, the function ignores the `cTable` parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you use this function on a computer that supports only basic QuickDraw, you may specify only `NIL` in this parameter.

aGDevice

A handle to a `GDevice` (IV-2264) structure that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case the function attaches this structure to the new offscreen graphics world. If you set the `pixelFormat` parameter to 0, or if you do not set the `noNewDevice` flag, the function ignores this parameter, so you should set it to `NIL`. If you set the `pixelFormat` parameter to 0, the function uses the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NIL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` structures for offscreen graphics worlds.

Q

QTNewGWorldFromPtr

`flags`

A constant (see below) that identifies an option available to your application. If you don't wish to use this option, pass 0 in this parameter. In this case the default behavior is to create an offscreen graphics world that uses an existing GDevice structure (if you pass 0 in the depth parameter) or creates a new GDevice structure. Most constants used in creating a GWorld are irrelevant for this function, as its purpose is to wrap a GWorld around an existing block of pixels rather than to define and create a pixmap.

`baseAddr`

The base address for the pixel data.

`rowBytes`

The total size of the pixel data divided by the height of the pixel map. In other words, the number of bytes in one row of pixels or the number of bytes between vertically adjacent pixels.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`noNewDevice`

Do not create an offscreen GDevice structure.

Discussion

This function wraps a GWorld around an existing pixel map. Note that it does not copy the pixmap. A subsequent call to `DisposeGWorld` will not dispose of the pixel map; it will only dispose of the GWorld wrapper. It is the caller's responsibility to dispose of the pixel map.

You can use this call to allocate an offscreen graphics world using special memory (such as on a video card). If you have an image in memory that belong to something else (a hardware screen buffer, a 3D card, or another file format or program), you can use this function to wrap a graphics world around the image and then use QuickTime calls on that graphics world to compress it, scale it, draw to it, and so on. If your new graphics world has a planar pixel format, you must use this call instead of `QTNewGWorld` (II–1204).

Special Considerations

Do not unlock the pixels of the allocated graphics world. If your original pixels are from another graphics world then you must ensure that the source pixels are locked.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

QTNewTween

Undocumented

```
OSErr QTNewTween (
    QTweener      *tween,
    QTAtomContainer container,
    QTAtom        tweenAtom,
    TimeValue     maxTime );
```

tween

A pointer to a pointer to a QTweenerRecord (IV–2391) structure.

container

Undocumented

tweenAtom

Undocumented

maxTime

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

QTNextChildAnyType

Returns the next atom in the child list of the specified parent atom.

```
OSErr QTNextChildAnyType (
    QTAtomContainer container,
    QTAtom        parentAtom,
    QTAtom        currentChild,
    QTAtom        *nextChild );
```

container

The atom container that contains the parent atom.

QTParseTextHREF

parentAtom

The parent atom for this operation.

currentChild

The last atom retrieved by this function. To retrieve the first atom in the child list, set the value of currentChild to 0.

nextChild

A pointer to an uninitialized QT atom data structure. On return, the data structure contains the offset of the next atom in the child list after the atom specified by currentChild, or 0 if the atom specified by currentChild was the last atom in the list.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You can call this function to iterate through all the atoms in a parent atom’s child list, regardless of their types and IDs.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Retrieving Atoms and Atom Data” (V–2768)

Related Java Methods

quicktime.std.movies.AtomContainer.nextChildAnyType()

QTParseTextHREF

Undocumented

```
OSErr QTParseTextHREF (
    char          *href,
    SInt32        hrefLen,
    QTAtomContainer inContainer,
    QTAtomContainer *outContainer );
```

href

A pointer to an HREF string.

hrefLen

The length of the HREF string.

inContainer

Undocumented

outContainer

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

QTPhotoDefineHuffmanTable

Defines a Huffman table.

```
ComponentResult QTPhotoDefineHuffmanTable (
    ComponentInstance    codec,
    short                componentNumber,
    Boolean              isDC,
    unsigned char        *lengthCounts,
    unsigned char        *values );
```

codec

Identifies your connection to the image compressor component.

componentNumber

Specifies a color component. If 0, the luminance Huffman table is set. If 1, the chrominance Huffman table is set.

isDC

If TRUE, the DC Huffman table is set. If FALSE, the AC Huffman table is set.

lengthCounts

A pointer to an array of 16 length counts.

values

A pointer to an array of Huffman values.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.



QTPhotoDefineQuantizationTable

Discussion

This function lets you define a Huffman table to be used in future JPEG compression operations. Normally the JPEG image compressor components use the default Huffman tables as specified in sections K.3 through K.6 of the JPEG specification. You can use this function to override the default tables.

Special Considerations

This call is supported only by the Photo JPEG and Motion JPEG compressors. Only advanced programmers will need to use this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

QTPhotoDefineQuantizationTable

Specifies a custom quantization table.

```
ComponentResult QTPhotoDefineQuantizationTable (  
    ComponentInstance    codec,  
    short                componentNumber,  
    unsigned char        *table );
```

codec

Identifies your connection to the image compressor component.

componentNumber

If 0, the luminance quantization table is set. If 1, the chrominance quantization table is set.

table

A pointer to an array of 64 quantization values.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

By default, the JPEG compressors select quantization tables based on quality settings. This function lets you override these tables with tables of your own choice.

Special Considerations

This call is only supported by the Photo JPEG and Motion JPEG compressors. Only advanced programmers will need to use this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

QTPhotoSetRestartInterval

Specifies the restart interval to use in future JPEG compression operations.

```
ComponentResult QTPhotoSetRestartInterval (
    ComponentInstance    codec,
    unsigned short       restartInterval );
```

codec

Identifies your connection to the image compressor component.

restartInterval

The new restart interval. Pass 0 to tell the compressor not to insert restart markers in the data stream.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

By default, the JPEG compressor components do not insert restart markers in the compressed data stream unless the “optimize for streaming” setting is selected.

Special Considerations

This call is supported only by the Photo JPEG and Motion JPEG compressors. Only advanced programmers will need to use this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCodec.h

QTPhotoSetSampling

Specifies the chrominance downsampling ratio to use in future JPEG compression operations.

```
ComponentResult QTPhotoSetSampling (
    ComponentInstance    codec,
    short                yH,
    short                yV,
    short                cbH,
    short                cbV,
```

QTRegisterAccessKey

```
short      crH,  
short      crV );
```

`codec`

Identifies your connection to the image compressor component.

`yH`

The number of horizontal luminance blocks to put in each macroblock.

`yV`

The number of vertical luminance blocks to put in each macroblock.

`cbH`

The number of horizontal chroma blue blocks to put in each macroblock.

`cbV`

The number of vertical chroma blue blocks to put in each macroblock.

`crH`

The number of horizontal chroma red blocks to put in each macroblock.

`crV`

The number of vertical chroma red blocks to put in each macroblock.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

By default, the Photo JPEG compressor uses 4:1:1 chroma downsampling and the Motion JPEG compressors use 4:2:2 chroma downsampling for most quality settings. For `codecLosslessQuality`, both compressors disable chroma downsampling. Currently the only supported downsampling ratios are none (pass 1,1,1,1,1,1), 4:2:2 (pass 2,1,1,1,1,1) and 4:1:1 (pass 2,2,1,1,1,1).

Special Considerations

This call is supported only by the Photo JPEG and Motion JPEG compressors. Only advanced programmers will need to use this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCodec.h`

QTRegisterAccessKey

Registers an access key.

```
OSErr QTRegisterAccessKey (
```

```
Str255    accessKeyType,
long      flags,
Handle    accessKey );
```

accessKeyType

The access key type of the key to be registered.

flags

Flags that specify the operation of this function. To register a system access key, set the `kAccessKeySystemFlag` flag (see below). To register an application access key, set this parameter to 0.

accessKey

A handle to the key to be registered.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error or if the access key has already been registered.

flags Constant

`kAccessKeySystemFlag`

Set to 1 to register a system access key.

Discussion

Most access keys are strings. A string stored in the `accessKey` handle does not include a trailing zero or leading length byte; to get the length of the string, get the size of the handle. If the access key has already been registered, no error is returned, and the request is simply ignored.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Registering and Unregistering Access Keys” (V–2771)

QTRemoveAtom

Removes an atom and its children from the specified atom container.

```
OSErr QTRemoveAtom (
    QAtomContainer  container,
    QAtom          atom );
```

container

The atom container for this operation. The atom container must not be locked.

QTRemoveChildren

atom

The atom to be removed from the container.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You call this function to remove a particular atom and its children from an atom container. To remove all the atoms in an atom container, you should use `QTRemoveChildren` (II–1216).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Removing Atoms From an Atom Container” (V–2770)

Related Java Methods

`quicktime.std.movies.AtomContainer.removeAtom()`

QTRemoveChildren

Removes all the children of an atom from the specified atom container.

```
OSErr QTRemoveChildren (
    QTAtomContainer  container,
    QTAtom           atom );
```

container

The atom container for this operation. The atom container must not be locked.

atom

The atom whose children should be removed. To remove all the atoms in the atom container, pass a value of `kParentAtomIsContainer`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Removing Atoms From an Atom Container” (V–2770)

Related Java Methods

`quicktime.std.movies.AtomContainer.removeChildren()`

QTReplaceAtom

Replaces the contents of an atom and its children with a different atom and its children.

```
OSErr QTReplaceAtom (
    QTAtomContainer    targetContainer,
    QTAtom             targetAtom,
    QTAtomContainer    replacementContainer,
    QTAtom             replacementAtom );
```

targetContainer

The atom container that contains the atom to be replaced. The atom container must not be locked.

targetAtom

The atom to be replaced.

replacementContainer

The atom container that contains the replacement atom.

replacementAtom

The replacement atom.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The target atom and the replacement atom must be of the same type. The target atom maintains its original atom ID. This function does not modify the replacement container.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Copying Existing Atoms” (V–2767)

QTSAAllocBuffer

Related Java Methods

`quicktime.std.movies.AtomContainer.replaceAtom()`

QTSAAllocBuffer

Allocates a QuickTime streaming stream buffer.

```
QTStreamBuffer * QTSAAllocBuffer (
    SInt32      size );
```

size

The size of the buffer to be allocated.

function result A QTStreamBuffer (IV–2385) structure

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeStreaming.h`

QTSAAllocMemPtr

Undocumented

```
QTSMemPtr QTSAAllocMemPtr (
    UInt32      inByteCount,
    SInt32      inFlags );
```

inByteCount

Undocumented

inFlags

Undocumented

function result *Undocumented*

inFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTScheduledBandwidthRelease

Undocumented

```
OSErr QTScheduledBandwidthRelease (
    QTScheduledBandwidthReference sbwRef,
    long flags );
```

sbwRef

A pointer to an opaque data structure.

flags

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

QTScheduledBandwidthRequest

Undocumented

```
OSErr QTScheduledBandwidthRequest (
    QTScheduledBandwidthPtr scheduleRec,
    QTBandwidthNotificationUPP notificationCallback,
    void *refcon,
    QTScheduledBandwidthReference *sbwRef,
    long flags );
```

scheduleRec

A pointer to a QTScheduledBandwidthRecord (IV-2366) structure.

QTSCopyMessage

notificationCallback

A **Universal Procedure Pointer** that accesses a QTBandwidthNotificationProc (III–2124) callback.

refcon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

sbwRef

A pointer to an opaque data structure.

flags

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

QTSCopyMessage

Undocumented

```
QTStreamBuffer * QTSCopyMessage (  
    QTStreamBuffer *inStreamBuffer );
```

inStreamBuffer

A pointer to a QTStreamBuffer (IV–2385) structure.

function result A pointer to a QTStreamBuffer (IV–2385) structure.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSDisposePresentation

Disposes of a QuickTime streaming presentation.

```
OSErr QTSDisposePresentation (
    QTSPresentation  inPresentation,
    SInt32           inFlags );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines the presentation to be disposed.

inFlags

Flags governing the disposal of the presentation. Currently, no flags are defined; set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSDisposeStatHelper

Disposes of a QuickTime streaming statistics helper that was previously created by QTNewStatHelper (II–1255).

```
OSErr QTSDisposeStatHelper (
    QTStatHelper  inStatHelper );
```

inStatHelper

A pointer to a QTStatHelperRecord (IV–2384) structure that defines the statistics helper to be disposed.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSDisposeStream

QTSDisposeStream

Disposes of a QuickTime streaming stream.

```
OSErr QTSDisposeStream (
    QTSSStream    inStream,
    SInt32        inFlags );
```

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream to be disposed.

inFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSDuplicateMessage

Undocumented

```
OSErr QTSDuplicateMessage (
    QTSSStreamBuffer *inMessage,
    SInt32           inFlags,
    QTSSStreamBuffer **outDuplicatedMessage );
```

inMessage

Undocumented

inFlags

Undocumented

outDuplicatedMessage

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSDupMessage

Undocumented

```
QTSSStreamBuffer * QTSDupMessage (
    QTSSStreamBuffer *inStreamBuffer );
```

inStreamBuffer

A pointer to a QTSSStreamBuffer (IV–2385) structure.

function result A pointer to a QTSSStreamBuffer (IV–2385) structure.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSetAtomData

Changes the data of a leaf atom.

```
OSErr QTSetAtomData (
    QTAtomContainer container,
    QTAtom atom,
    long dataSize,
    void *atomData );
```

container

The atom container that contains the atom to be modified.

atom

The atom to be modified.

dataSize

The length, in bytes, of the data pointed to by the atomData parameter.

atomData

A pointer to the new data for the atom.



QTSetAtomData

function result Only leaf atoms contain data; this function returns an error if you pass it to a nonleaf atom. You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

You call this function to replace a leaf atom’s data with new data. The atom container specified by the `container` parameter should not be locked. The following code illustrates using this function to update an atom container that describes a **sprite**:

```
// QTSetAtomData coding example
OSErr SetSpriteData (QTAtomContainer sprite, Point *location,
    short *visible, short *layer, short *imageIndex)
{
    OSErr err = noErr;
    QTAtom propertyAtom;

    // if the sprite’s visible property has a new value
    if (visible)
    {
        // retrieve the atom for the visible property
        // -- if none exists, insert one
        if ((propertyAtom = QTFindChildByIndex (sprite,
            kParentAtomIsContainer, kSpritePropertyVisible, 1,
            NIL)) == 0)
            FailOSErr (QTInsertChild (sprite, kParentAtomIsContainer,
                kSpritePropertyVisible, 1, 1, sizeof(short), visible,
                NIL));

        // if an atom does exist, update its data
        else
            FailOSErr (QTSetAtomData (sprite, propertyAtom,
                sizeof(short), visible));
    }
}
```

Special Considerations

This function may move memory; if the pointer specified by the `atomData` parameter is a dereferenced handle, you should lock the handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`
 Programming summary: “Modifying Atoms” (V–2769)

Related Java Methods

`quicktime.std.movies.AtomContainer.setAtomData()`

QTSetAtomID

Changes the ID of an atom.

```
OSErr QTSetAtomID (
    QTAtomContainer    container,
    QTAtom             atom,
    QTAtomID           newID );
```

container

The atom container for this operation.

atom

The atom to be modified. You cannot change the ID of the container by passing 0 for the *atom* parameter.

newID

The new ID for the atom. You cannot change an atom’s ID to an ID already assigned to a sibling atom of the same type.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`
 Programming summary: “Modifying Atoms” (V–2769)

Related Java Methods

`quicktime.std.movies.AtomContainer.setAtomID()`

QTSetDDObject

Sets the Windows DirectDraw object currently in use by QuickTime.



QTSetDDPrimarySurface

```
OSErr QTSetDDObject (
    void      *lpNewDDObject );
```

lpNewDDObject

The DirectDraw object.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function is useful for developers who want to call Windows DirectDraw methods directly.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

See Also

For the corresponding get function, see QTGetDDObject (II–1174).

QTSetDDPrimarySurface

Sets the primary Windows DirectDraw surface used by QuickTime.

```
OSErr QTSetDDPrimarySurface (
    void      *lpNewDDSurface,
    unsigned long  flags );
```

lpNewDDSurface

A pointer to a DirectDraw surface.

flags

Contains flags (see below) that control the set operation.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

flags Constants

kDDSurfaceLocked

If set, QuickTime won’t attempt to lock the graphics device when blitting to the PixMap.

kDDSurfaceStatic

If set, QuickTime assumes that windows on this device do not move.

Discussion

This function is useful for multimedia developers who want to wrap QuickTime around surfaces they have already created.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

QTSetPixMapHandleGammaLevel

Sets the gamma level of a pixel map.

```
OSErr QTSetPixMapHandleGammaLevel (
    PixMapHandle    pm,
    Fixed            gammaLevel);
```

pm

A handle to a `PixMap` (IV–2332) structure that has a `PixMapExtension` (IV–2335) structure.

gammaLevel

The new gamma level.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the gamma level of a pixel map before compressing it so that the codec knows if it needs to do additional gamma correcting when compressing.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding get function, see `QTGetPixMapHandleGammaLevel` (II–1180).

QTSetPixMapHandleRequestedGammaLevel

Sets the requested gamma level of a pixel map.

QTSetPixMapHandleRowBytes

```
OSErr QTSetPixMapHandleRequestedGammaLevel (
    PixMapHandle    pm,
    Fixed            requestedGammaLevel);
```

pm

A handle to a `PixMap` (IV–2332) structure that has a `PixMapExtension` (IV–2335) structure.

requestedGammaLevel

A specified gamma level or a constant (see below).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

requestedGammaLevel Constants

`kQTUsePlatformDefaultGammaLevel`

The default gamma level for the platform (Macintosh is 1.8, Windows is 2.5).

`kQTUseSourceGammaLevel`

Leave the pixel map with the gamma level of the source data, instead of converting it.

`kQTCCIR601VideoGammaLevel`

Gamma 2.2, for ITU-R BT.601 based video.

Discussion

This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the requested gamma level of a pixel map before decompressing so that the codec knows what gamma correction is necessary when decompressing into the `PixMap` structure. The resulting gamma level can then be found by calling `QTGetPixMapHandleGammaLevel` (II–1180).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding get function, see `QTGetPixMapHandleRequestedGammaLevel` (II–1180).

QTSetPixMapHandleRowBytes

Sets the `rowBytes` value for a pixel map accessed by a handle.

```
OSErr QTSetPixMapHandleRowBytes (
    PixMapHandle    pm,
```

```
long          rowBytes );
```

pm

A handle to a `PixMap` (IV–2332) structure.

rowBytes

The `rowBytes` value to be set.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding get function, see `QTGetPixMapHandleRowBytes` (II–1181).

QTSetPixMapPtrGammaLevel

Sets the gamma level of a pixel map.

```
OSErr QTSetPixMapPtrGammaLevel (
    PixMapPtr    pm,
    Fixed         gammaLevel);
```

pm

A pointer to a `PixMap` (IV–2332) structure that has a `PixMapExtension` (IV–2335) structure.

gammaLevel

The new gamma level.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the gamma level of a pixel map before compressing it so that the codec knows if it needs to do additional gamma correcting when compressing.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCompression.h`

QTSetPixMapPtrRequestedGammaLevel

See Also

For the corresponding get function, see QTGetPixMapPtrGammaLevel (II–1181).

QTSetPixMapPtrRequestedGammaLevel

Sets the requested gamma level of a pixel map.

```
OSErr QTSetPixMapPtrRequestedGammaLevel (
    PixMapPtr    pm,
    Fixed         requestedGammaLevel);
```

pm

A pointer to a `PixMap` (IV–2332) structure that has a `PixMapExtension` (IV–2335) structure.

requestedGammaLevel

A specified gamma level or a constant (see below).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

requestedGammaLevel Constants

`kQTUsePlatformDefaultGammaLevel`

The default gamma level for the platform (Macintosh is 1.8, Windows is 2.5).

`kQTUseSourceGammaLevel`

Leave the pixel map with the gamma level of the source data, instead of converting it.

`kQTCCIR601VideoGammaLevel`

Gamma 2.2, for ITU-R BT.601 based video.

Discussion

This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the requested gamma level of a pixel map before decompressing so that the codec knows what gamma correction is necessary when decompressing into the `PixMap` structure. The resulting gamma level can then be found by calling `QTGetPixMapPtrGammaLevel` (II–1181).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding get function, see `QTGetPixMapPtrRequestedGammaLevel` (II–1182).

QTSetPixMapPtrRowBytes

Sets the `rowBytes` value for a pixel map accessed by a pointer.

```
OSErr QTSetPixMapPtrRowBytes (
    PixMapPtr    pm,
    long         rowBytes );
```

`pm`

A pointer to a `PixMap` (IV–2332) structure.

`rowBytes`

The `rowBytes` value to be set.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

See Also

For the corresponding get function, see `QTGetPixMapPtrRowBytes` (II–1183).

QTSFindMediaPacketizer

Creates a list of media packetizers that can work with a specified sample description and meet specified criteria.

```
OSErr QTSFindMediaPacketizer (
    MediaPacketizerRequirementsPtr  inPacketizerinfo,
    SampleDescriptionHandle          inSampleDescription,
    RTPPayloadSortRequestPtr        inSortInfo,
    QTAtomContainer                  *outPacketizerList );
```

`inPacketizerinfo`

A pointer to a `MediaPacketizerRequirements` (IV–2308) structure that specifies the required features of the media packetizers you are looking for.

`inSampleDescription`

A handle to a `SampleDescription` (IV–2414) structure that specifies the media data the packetizer needs to work with.

QTSFindMediaPacketizerForPayloadID

inSortInfo

A pointer to a `RTTTPayloadSortRequest` (IV–2410) structure that specifies the sort order for the list of packetizers.

outPacketizerList

On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available media packetizers that meet the specified criteria. Only packetizers that have the features specified by *inPacketizerInfo* will be listed. The list will be sorted in the order specified by *inSortInfo*.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

QTSFindMediaPacketizerForPayloadID

Creates a list of media packetizers for a specified payload number.

```
OSErr QTSFindMediaPacketizerForPayloadID (
    long                payloadID,
    RTTTPayloadSortRequestPtr  inSortInfo,
    QTAtomContainer     *outPacketizerList );
```

payloadID

An IETF payload number.

inSortInfo

A pointer to a `RTTTPayloadSortRequest` (IV–2410) structure that specifies the sort order for the list of packetizers.

outPacketizerList

On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available media packetizers for the specified payload ID. The list will be sorted in the order specified by *inSortInfo*.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

QTSTFindMediaPacketizerForPayloadName

Creates a list of media packetizers for a specified payload name.

```
OSErr QTSTFindMediaPacketizerForPayloadName (
    const char          *payloadName,
    RTPPayloadSortRequestPtr  inSortInfo,
    QTAtomContainer      *outPacketizerList );
```

payloadName

A pointer to a payload name string.

inSortInfo

A pointer to a RTPPayloadSortRequest (IV–2410) structure that specifies the sort order for the list of packetizers.

outPacketizerList

On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available media packetizers for the specified payload name. The list will be sorted in the order specified by *inSortInfo*.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

QTSTFindMediaPacketizerForTrack

Creates a list of media packetizers for a specified movie track and sample data.

```
OSErr QTSTFindMediaPacketizerForTrack (
    Track          inTrack,
    long           inSampleDescriptionIndex,
    RTPPayloadSortRequestPtr  inSortInfo,
    QTAtomContainer      *outPacketizerList );
```

QTSFindReassemblerForPayloadID

inTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

inSampleDescriptionIndex

The value of the `dataRefIndex` field of the `SampleDescription` (IV–2414) structure that specifies the type of media data that will be packetized.

inSortInfo

A pointer to a `RTPPayloadSortRequest` (IV–2410) structure that specifies the sort order for the list of packetizers.

outPacketizerList

On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available media packetizers for the specified track. The list will be sorted in the order specified by *inSortInfo*.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

QTSFindReassemblerForPayloadID

Creates a list of streaming reassemblers for a specified payload number.

```
OSErr QTSFindReassemblerForPayloadID (
    UInt8          inPayloadID,
    RTPPayloadSortRequest *inSortInfo,
    QTAtomContainer *outReassemblerList );
```

inPayloadID

An IETF payload number.

inSortInfo

A pointer to a `RTPPayloadSortRequest` (IV–2410) structure that specifies the sort order for the list of reassemblers.

outReassemblerList

On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available reassemblers for the specified track. The list will be sorted in the order specified by *inSortInfo*.

QTSFindReassemblerForPayloadName

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

QTSFindReassemblerForPayloadName

Creates a list of streaming reassemblers for a specified payload name.

```
OSErr QTSFindReassemblerForPayloadName (
    const char          *inPayloadName,
    RTPPayloadSortRequest *inSortInfo,
    QTAtomContainer      *outReassemblerList );
```

inPayloadName

A payload name string.

inSortInfo

A pointer to a RTPPayloadSortRequest (IV–2410) structure that specifies the sort order for the list of reassemblers.

outReassemblerList

On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available reassemblers for the specified track. The list will be sorted in the order specified by *inSortInfo*.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

QTSFlattenMessage

Undocumented

```
QTSSStreamBuffer * QTSFlattenMessage (
    QTSSStreamBuffer *inStreamBuffer );
```

QTSTFreeMessage

inStreamBuffer

A pointer to a QTStreamBuffer (IV–2385) structure.

function result A pointer to a QTStreamBuffer (IV–2385) structure.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSTFreeMessage

Undocumented

```
void QTSTFreeMessage (
    QTStreamBuffer    *inStreamBuffer );
```

inStreamBuffer

A pointer to a QTStreamBuffer (IV–2385) structure.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSGetErrorString

Undocumented

```
Boolean QTSGetErrorString (
    SInt32    inErrorCode,
    UInt32    inMaxErrorStringLength,
    char      *outErrorString,
    SInt32    inFlags );
```

inErrorCode

Undocumented

inMaxErrorStringLength

Undocumented

outErrorString

Undocumented

inFlags

Undocumented

function result Undocumented

inFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSGetNetworkAppName

Gets the name of a streaming network application.

```
OSErr QTSGetNetworkAppName (
    SInt32    inFlags,
    char      **outCStringPtr );
```

inFlags

A flag (see below) that determines whether the application name is a full pathname.

outCStringPtr

A Ptr to a CStringPtr; see MacTypes.h. This information is sent back to servers in HTTP and RTSP headers, so they can work out client statistics. A typical default string is QTS (qtver=4.1.1;cpu=PPC;os=Mac 9.0.4).

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

inFlags Constants

kQTSNetworkAppNameIsFullNameFlag

The application name is a full pathname.

Discussion

Following is an example of calling this function:

```
Ptr networkAppName = NIL;
err = QTSGetNetworkAppName(0L, &networkAppName);
printf("The NetworkAppName is %s", networkAppName);
DisposePtr(networkAppName);
```

QTSGetOrMakeStatAtomForStream

```
// This call prints
// The NetworkAppName is QTS (qtver=4.1.1;cpu=PPC;os=Mac 9.0.4)
// or
// The NetworkAppName is QTS (qtver=4.0;os=Windows NT 4.0 Service Pack 3)
// If you set it from your app, that will be returned instead.
```

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSSetNetworkAppName (II-1305).

QTSGetOrMakeStatAtomForStream

Gets the statistics atom for a stream or creates a new statistics atom for it.

```
OSErr QTSGetOrMakeStatAtomForStream (
    QTAtomContainer    inContainer,
    QTSSStream         inStream,
    QTAtom             *outParentAtom );
```

inContainer

An atom container that holds the statistics atoms for the specified stream.

inStream

A pointer to a QTSSStreamRecord (IV-2387) structure that defines a stream.

outParentAtom

On entry, a pointer to a variable of type QTAtom; on return, this variable is set to the atom that holds the statistics for this stream. If no such atom exists for that stream, then the function creates a statistics atom.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Special Considerations

This function is to be used only by stream components to put stream statistics into an atom container; applications should not call it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTStreamPresentation

Gets the presentation for a stream.

```
QTStreamPresentation QTStreamPresentation (
    QTStream    inStream );
```

inStream

A pointer to a QTStreamRecord (IV–2387) structure that defines a stream.

function result A pointer to a QTStreamPresentationRecord (IV–2379) structure.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTStreamInitializeMediaParams

Undocumented

```
OSStatus QTStreamInitializeMediaParams (
    QTStreamMediaParams    *inMediaParams );
```

inMediaParams

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTStreamInsertStatistic

Inserts statistics data into the statistic atom for a stream.

```
OSStatus QTStreamInsertStatistic (
    QTAtomContainer    inContainer,
    QTAtom             inParentAtom,
    OSType              inStatType,
```

QTSSInsertStatistic

```
void          *inStatData,  
UInt32       inStatDataLength,  
OSType       inStatDataFormat,  
SInt32       inFlags );
```

inContainer

A handle to the atom container that contains the statistic atom.

inParentAtom

The atom that will hold a new atom containing the specified statistic data.

inStatType

A constant (see below) that identifies the type of statistic atom to insert the data into.

inStatData

A pointer to a structure containing the data to insert.

inStatDataLength

The length, in bytes, of the statistic data.

inStatDataFormat

A constant (see below) that identifies the format of the inserted statistic atom.

inFlags

Currently no flags are defined; pass 0 in this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inStatType Constants

`kQTSSStatisticsStreamAtomType`

Stream atom type; value is 'strm'.

`kQTSSStatisticsNameAtomType`

Name atom type; value is 'name'.

`kQTSSStatisticsDataFormatAtomType`

Data format atom type; value is 'frmt'.

`kQTSSStatisticsDataAtomType`

Data atom type; value is 'data'.

`kQTSSStatisticsUnitsAtomType`

Units atom type; value is 'unit'.

`kQTSSStatisticsUnitsNameAtomType`

Units name atom type; value is 'unin'.

inStatDataFormat Constants

`kQTSSStatisticsSInt32DataFormat`

Signed 32-bit integer format; value is 'si32'.

kQTStatisticsUInt32DataFormat
 Unsigned 32-bit integer format; value is 'ui32'.
 kQTStatisticsSInt16DataFormat
 Signed 16-bit integer format; value is 'si16'.
 kQTStatisticsUInt16DataFormat
 Unsigned 16-bit integer format; value is 'ui16'.
 kQTStatisticsFixedDataFormat
 Fixed integer numeric format; value is 'fixd'.
 kQTStatisticsStringDataFormat
 String format; value is 'strg'.
 kQTStatisticsOSTypeDataFormat
 OSType (32-bit) format; value is 'ostp'.

Special Considerations

This function is to be used only by stream components to put stream statistics into an atom container; applications should not call it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTInsertStatisticName

Inserts the name and type of a statistic datum into the statistic atom for a stream.

```
OSErr QTInsertStatisticName (
    QTAtomContainer    inContainer,
    QTAtom             inParentAtom,
    OSType             inStatType,
    const char         *inStatName,
    UInt32             inStatNameLength );
```

inContainer

A handle to the atom container that contains the statistic atom. Both the atom container and the parent atom must already exist.

inParentAtom

The atom that will hold a new atom containing the specified statistic name and type.

QTSSInsertStatisticUnits

`inStatType`

A constant (see below) that identifies the type of statistic atom to insert the data into.

`inStatName`

A pointer to the name string to be inserted.

`inStatNameLength`

The length of the name string in characters.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inStatType Constants

`kQTSSStatisticsStreamAtomType`

Stream atom type; value is 'strm'.

`kQTSSStatisticsNameAtomType`

Name atom type; value is 'name'.

`kQTSSStatisticsDataFormatAtomType`

Data format atom type; value is 'frmt'.

`kQTSSStatisticsDataAtomType`

Data atom type; value is 'data'.

`kQTSSStatisticsUnitsAtomType`

Units atom type; value is 'unit'.

`kQTSSStatisticsUnitsNameAtomType`

Units name atom type; value is 'unin'.

Special Considerations

This function is to be used only by stream components to put stream statistics into an atom container; applications should not call it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeStreaming.h`

QTSSInsertStatisticUnits

Inserts the name and type of statistic units into the statistic atom for a stream.

```
OSErr QTSSInsertStatisticUnits (
    QTAtomContainer    inContainer,
    QTAtom             inParentAtom,
    OSType              inStatType,
```



```
OSType          inUnitsType,
const char      *inUnitsName,
UInt32          inUnitsNameLength );
```

inContainer

A handle to the atom container that contains the statistic atom. Both the atom container and the parent atom must already exist.

inParentAtom

The atom that will hold a new atom containing the specified statistic name and type.

inStatType

A constant (see below) that identifies the type of statistic atom to insert the data into.

inUnitsType

A constant (see below) that identifies the type of units atom to insert the data into.

inUnitsName

A pointer to the units name string to be inserted.

inUnitsNameLength

The length of the units name string in characters.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inStatType Constants

kQTSStatisticsStreamAtomType

Stream atom type; value is 'strm'.

kQTSStatisticsNameAtomType

Name atom type; value is 'name'.

kQTSStatisticsDataFormatAtomType

Data format atom type; value is 'frmt'.

kQTSStatisticsDataAtomType

Data atom type; value is 'data'.

kQTSStatisticsUnitsAtomType

Units atom type; value is 'unit'.

kQTSStatisticsUnitsNameAtomType

Units name atom type; value is 'unin'.

inUnitsType Constants

kQTSStatisticsNoUnitsType

No unit type; value is 0.

QTSMediaGetIndStreamInfo

kQTStatisticsPercentUnitsType

Percent unit type; value is 'pcnt'.

kQTStatisticsBitsPerSecUnitsType

Bits per second unit type; value is 'bps '.

kQTStatisticsFramesPerSecUnitsType

Frames-per-second unit type; value is 'fps '.

Special Considerations

This function is to be used only by stream components to put stream statistics into an atom container; applications should not call it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSMediaGetIndStreamInfo

Undocumented

```
ComponentResult QTSMediaGetIndStreamInfo (
    MediaHandler    mh,
    SInt32          inIndex,
    OSType          inSelector,
    void            *ioParams );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I-432).

inIndex

Undocumented

inSelector

A constant (see below) that identifies the type of information to be retrieved.

ioParams

A pointer to returned information in a format determined by inSelector (see below).

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

inSelector Constants

kQTSMediaPresentationInfo

The `ioParams` parameter returns a `QTSMediaPresentationParams` (IV–2375) structure identifying the presentation. Constant value is 'pres'.

kQTSMediaNotificationInfo

The `ioParams` parameter returns a `QTSMediaNotificationParams` (IV–2374) structure identifying the notification process. Constant value is 'noti'.

kQTSMediaTotalDataRateInfo

The `ioParams` parameter returns a `UInt32` value representing bits per second. Constant value is 'dtrt'.

kQTSMediaLostPercentInfo

The `ioParams` parameter returns a fixed integer value representing the percentage of data lost. Constant value is 'lspc'.

kQTSMediaNumStreamsInfo

The `ioParams` parameter returns a `UInt32` value representing the number of streams. Constant value is 'nstr'.

kQTSMediaIndSampleDescriptionInfo

The `ioParams` parameter returns a `QTSMediaIndSampleDescriptionParams` (IV–2373) structure identifying the sample data. Constant value is 'isdc'.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTSMovie.h`

See Also

For the corresponding set function, see `QTSMediaSetIndStreamInfo` (II–1246).

QTSMediaGetInfo

Gets information about a streaming media.

```
ComponentResult QTSMediaGetInfo (
    MediaHandler      mh,
    OSType             inSelector,
    void               *ioParams );
```

`mh`

A media handler. You can obtain this reference from `GetMediaHandler` (I–432).

QTSMediaSetIndStreamInfo

inSelector

A constant (see below) that identifies the type of information to be retrieved.

ioParams

A pointer to returned information in a format determined by *inSelector* (see below).

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

inSelector Constants

kQTSMediaPresentationInfo

The *ioParams* parameter returns a *QTSMediaPresentationParams* (IV–2375) structure identifying the presentation. Constant value is 'pres'.

kQTSMediaNotificationInfo

The *ioParams* parameter returns a *QTSMediaNotificationParams* (IV–2374) structure identifying the notification process. Constant value is 'noti'.

kQTSMediaTotalDataRateInfo

The *ioParams* parameter returns a *UInt32* value representing bits per second. Constant value is 'dtrt'.

kQTSMediaLostPercentInfo

The *ioParams* parameter returns a fixed integer value representing the percentage of data lost. Constant value is 'lspc'.

kQTSMediaNumStreamsInfo

The *ioParams* parameter returns a *UInt32* value representing the number of streams. Constant value is 'nstr'.

kQTSMediaIndSampleDescriptionInfo

The *ioParams* parameter returns a *QTSMediaIndSampleDescriptionParams* (IV–2373) structure identifying the sample data. Constant value is 'isdc'.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: *QTSMovie.h*

See Also

For the corresponding set function, see *QTSMediaSetInfo* (II–1248).

QTSMediaSetIndStreamInfo

Undocumented

```
ComponentResult QTSMediaSetIndStreamInfo (
    MediaHandler      mh,
    SInt32            inIndex,
    OSType            inSelector,
    void              *ioParams );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

inIndex

Undocumented

inSelector

A constant (see below) that identifies the type of information to be set.

ioParams

A pointer to information in a format determined by inSelector (see below).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inSelector Constants

kQTSMediaPresentationInfo

The ioParams parameter returns a QTSMediaPresentationParams (IV–2375) structure identifying the presentation. Constant value is 'pres'.

kQTSMediaNotificationInfo

The ioParams parameter returns a QTSMediaNotificationParams (IV–2374) structure identifying the notification process. Constant value is 'noti'.

kQTSMediaTotalDataRateInfo

The ioParams parameter returns a UInt32 value representing bits per second. Constant value is 'dtrt'.

kQTSMediaLostPercentInfo

The ioParams parameter returns a fixed integer value representing the percentage of data lost. Constant value is 'lspc'.

kQTSMediaNumStreamsInfo

The ioParams parameter returns a UInt32 value representing the number of streams. Constant value is 'nstr'.

kQTSMediaIndSampleDescriptionInfo

The ioParams parameter returns a QTSMediaIndSampleDescriptionParams (IV–2373) structure identifying the sample data. Constant value is 'isdc'.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTSMovie.h

QTSMediaSetInfo

See Also

For the corresponding get function, see QTSMediaGetIndStreamInfo (II–1244).

QTSMediaSetInfo

Sets information about a streaming media.

```
ComponentResult QTSMediaSetInfo (  
    MediaHandler    mh,  
    OSType          inSelector,  
    void            *ioParams );
```

mh

A media handler. You can obtain this reference from GetMediaHandler (I–432).

inSelector

A constant (see below) that identifies the type of information to be set.

ioParams

A pointer to information in a format determined by *inSelector* (see below).

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

inSelector Constants

kQTSMediaPresentationInfo

The *ioParams* parameter returns a QTSMediaPresentationParams (IV–2375) structure identifying the presentation. Constant value is 'pres'.

kQTSMediaNotificationInfo

The *ioParams* parameter returns a QTSMediaNotificationParams (IV–2374) structure identifying the notification process. Constant value is 'noti'.

kQTSMediaTotalDataRateInfo

The *ioParams* parameter returns a UInt32 value representing bits per second. Constant value is 'dtrt'.

kQTSMediaLostPercentInfo

The *ioParams* parameter returns a fixed integer value representing the percentage of data lost. Constant value is 'lspc'.

kQTSMediaNumStreamsInfo

The *ioParams* parameter returns a UInt32 value representing the number of streams. Constant value is 'nstr'.

kQTSMediaIndSampleDescriptionInfo

The *ioParams* parameter returns a QTSMediaIndSampleDescriptionParams (IV–2373) structure identifying the sample data. Constant value is 'isdc'.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTSMovie.h

See Also

For the corresponding get function, see QTSMediaGetInfo (II–1245).

QTSMMessageLength

Undocumented

```
SInt32 QTSMMessageLength (
    QTSSStreamBuffer *inStreamBuffer );
```

inStreamBuffer

A pointer to a QTSSStreamBuffer (IV–2385) structure.

function result The message length.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNewHandle

Allocates a new handle for data, with options and checking.

```
Handle QTSNewHandle (
    UInt32    inByteCount,
    SInt32    inFlags,
    SInt32    *outFlags );
```

inByteCount

The requested size in bytes of the relocatable block.

inFlags

Flags (see below) that control memory allocation options.

outFlags

A pointer to memory where return flags (see below) report on the block's actual memory location.

QTSNewPresentation

function result The new handle.

inFlags Constants

kQTSMemAllocClearMem

Set all bits in the allocated block to 0.

kQTSMemAllocDontUseTempMem

Don't allocate the block in temporary memory.

kQTSMemAllocTryTempMemFirst

Try first to allocate the block in temporary memory.

kQTSMemAllocDontUseSystemMem

Don't allocate the block in system memory.

kQTSMemAllocTrySystemMemFirst

Try first to allocate the block in system memory.

kQTSMemAllocHoldMemory

Make the block resident in physical memory and ineligible for paging.

kQTSMemAllocIsInterruptTime

This function is being called during an interrupt.

outFlags Constants

kQTSMemAllocAllocatedInTempMem

The block was allocated in temporary memory.

kQTSMemAllocAllocatedInSystemMem

The block was allocated in system memory.

Discussion

This function is a handy way to allocate memory without overflowing the application heap, which is mostly a concern with Mac OS versions 7 through 9. It is often used for streaming data.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNewPresentation

Creates a new streaming presentation.

```
OSErr QTSNewPresentation (
    const QTSNewPresentationParams    *inParams,
    QTSPresentation                    *outPresentation );
```


inParams

A pointer to a QTSNewPresentationParams (IV–2376) structure that specifies the presentation.

outPresentation

A pointer to a pointer to a new QTSPresentationRecord (IV–2379) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNewPresentationFromData

Undocumented

```
OSErr QTSNewPresentationFromData (
    OSType          inDataType,
    const void      *inData,
    const SInt64    *inDataLength,
    const QTSPresParams *inPresParams,
    QTSPresentation *outPresentation );
```

inDataType

Undocumented

inData

Undocumented

inDataLength

Undocumented

inPresParams

Undocumented

outPresentation

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h



QTSNewPresentationFromDataRef

QTSNewPresentationFromDataRef

Undocumented

```
OSErr QTSNewPresentationFromDataRef (
    Handle          inDataRef,
    OSType          inDataRefType,
    const QTSPresParams *inPresParams,
    QTSPresentation *outPresentation );
```

inDataRef

Undocumented

inDataRefType

Undocumented

inPresParams

Undocumented

outPresentation

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNewPresentationFromFile

Undocumented

```
OSErr QTSNewPresentationFromFile (
    const FSSpec      *inFileSpec,
    const QTSPresParams *inPresParams,
    QTSPresentation *outPresentation );
```

inFileSpec

Undocumented

inPresParams

Undocumented

outPresentation

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNewPtr

Allocates a block of memory for streaming data, with options and checking, and returns a pointer to it.

```
Ptr QTSNewPtr (
    UInt32    inByteCount,
    SInt32    inFlags,
    SInt32    *outFlags );
```

inByteCount

The requested size in bytes of the new memory block.

inFlags

Flags (see below) that control memory allocation options.

outFlags

A pointer to memory where return flags (see below) report on the block’s actual memory location.

function result A pointer to the newly allocated block.

inFlags Constants

kQTSMemAllocClearMem

Set all bits in the allocated block to 0.

kQTSMemAllocDontUseTempMem

Don’t allocate the block in temporary memory.

kQTSMemAllocTryTempMemFirst

Try first to allocate the block in temporary memory.

kQTSMemAllocDontUseSystemMem

Don’t allocate the block in system memory.

kQTSMemAllocTrySystemMemFirst

Try first to allocate the block in system memory.

kQTSMemAllocHoldMemory

Make the block resident in physical memory and ineligible for paging.

QTSNewSourcer

kQTSMemAllocIsInterruptTime

This function is being called during an interrupt.

outFlags Constants

kQTSMemAllocAllocatedInTempMem

The block was allocated in temporary memory.

kQTSMemAllocAllocatedInSystemMem

The block was allocated in system memory.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNewSourcer

Undocumented

```
OSErr QTSNewSourcer (
    void                *params,
    const QTSSourcerInitParams *inInitParams,
    SInt32              inFlags,
    ComponentInstance    *outSourcer );
```

params

Undocumented

inInitParams

Undocumented

inFlags

Undocumented

outSourcer

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

QTSNewStatHelper

Creates a new statistics helper for a stream or presentation.

```
OSErr QTSNewStatHelper (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
    OSType              inStatType,
    SInt32              inFlags,
    QTSSStatHelper      *outStatHelper );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines the presentation to keep statistics on. To create a statistics helper for a particular stream, pass in kQTSInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines the stream to keep statistics on. To create a statistics helper for a whole presentation, pass in kQTSAllStreams.

inStatType

A constant (see below) that defines the type of statistic you want the statistics helper to gather.

inFlags

Constants (see below) governing the action of the statistics helper.

outStatHelper

On entry, a pointer to a variable of type QTSSStatHelper; on return, this variable is set to the new statistics helper.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inStatType Constants

kQTSAllStatisticsType

Create a full statistics helper for all statistics; constant value is 'all'.

kQTSShortStatisticsType

Create a short statistics helper; constant value is 'shrt'.

kQTSSummaryStatisticsType

Create a summary statistics helper; constant value is 'summ'.

inFlags Constants

kQTSGetNameStatisticsFlag

The statistics helper is to get name statistics.

QTSNewStreamBuffer

kQTSDontGetDataStatisticsFlag

The statistics helper is to get data statistics.

kQTSUpdateAtomsStatisticsFlag

The statistics helper is to update statistics atoms.

kQTSGetUnitsStatisticsFlag

The statistics helper is to get units statistics.

Discussion

A statistics helper is a set of utility functions that you can use to retrieve and parse statistics from a stream component. You need to instantiate a statistics helper for every stream from which you want to gather statistics.

Special Considerations

When you are done using the statistics helper, call QTSDisposeStatHelper (II–1221).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNewStreamBuffer

Undocumented

```
OSErr QTSNewStreamBuffer (
    UInt32          inDataSize,
    SInt32          inFlags,
    QTSSStreamBuffer **outStreamBuffer );
```

inDataSize

Undocumented

inFlags

Undocumented

outStreamBuffer

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPrefsAddConnectionSetting

Undocumented

```
OSErr QTSPrefsAddConnectionSetting (
    OSType    protocol,
    SInt32    portID,
    UInt32    flags,
    UInt32    seed );
```

protocol

A constant (see below) that identifies the connection protocol.

portID

Undocumented

flags

Undocumented

seed

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

protocol Constants

kQTSDirectConnectHTTPProtocol

HTTP protocol; constant value is 'http'.

kQTSDirectConnectRTSPProtocol

RTSP protocol; constant value is 'rtsp'.

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPrefsAddProxySetting

Undocumented

```
OSErr QTSPrefsAddProxySetting (
    OSType    proxyType,
```

QTSPrefsAddProxyUserInfo

```
SInt32    portID,  
UInt32    flags,  
UInt32    seed,  
Str255    srvrURL );
```

proxyType

A constant (see below) that defines the proxy type.

portID

Undocumented

flags

Undocumented

seed

Undocumented

srvrURL

A string containing the server's URL.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

proxyType Constants

kQTSHHTTPProxyPrefsType

HTTP proxy; constant value is 'http'.

kQTSRTSPProxyPrefsType

RTSP proxy; constant value is 'rtsp'.

kQTSSOCKSProxyPrefsType

SOCKS proxy; constant value is 'scks'.

kQTSDontProxyDataType

No proxy; constant value is 'data'.

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPrefsAddProxyUserInfo

Undocumented


```
OSErr QTSPrefsAddProxyUserInfo (
    OSType      proxyType,
    SInt32      flags,
    SInt32      flagsMask,
    StringPtr   username,
    StringPtr   password );
```

proxyType

Undocumented

flags

Undocumented

flagsMask

Undocumented

username

Undocumented

password

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPrefsFindConnectionByType

Undocumented

```
OSErr QTSPrefsFindConnectionByType (
    OSType      protocol,
    UInt32      flags,
    UInt32      flagsMask,
    QTSTransportPref **connectionHndl,
    SInt16      *count );
```

protocol

A constant (see below) that identifies the connection protocol.

flags

Undocumented

QTSPrefsFindProxyByType

flagsMask

Undocumented

connectionHnd1

A handle to a QTSTransportPref (IV–2388) structure.

count

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

protocol Constants

kQTSDirectConnectHTTPProtocol

HTTP protocol; constant value is 'http'.

kQTSDirectConnectRTSPProtocol

RTSP protocol; constant value is 'rtsp'.

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPrefsFindProxyByType

Undocumented

```
OSErr QTSPrefsFindProxyByType (
    OSType          proxyType,
    UInt32          flags,
    UInt32          flagsMask,
    QTSProxyPref    **proxyHnd1,
    SInt16          *count );
```

proxyType

A constant (see below) that defines the proxy type.

flags

Undocumented

flagsMask

Undocumented

proxyHnd1

A handle to a QTSProxyPref (IV–2380) structure.

count

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

proxyType Constants

kQTSHHTTPProxyPrefsType

HTTP proxy; constant value is 'http'.

kQTSRTSPProxyPrefsType

RTSP proxy; constant value is 'rtsp'.

kQTSSOCKSPProxyPrefsType

SOCKS proxy; constant value is 'scks'.

kQTSDontProxyDataType

No proxy; constant value is 'data'.

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPrefsFindProxyUserInfoByType

Undocumented

```
OSErr QTSPrefsFindProxyUserInfoByType (
    OSType      proxyType,
    SInt32      flags,
    SInt32      flagsMask,
    StringPtr   username,
    StringPtr   password );
```

proxyType

Undocumented

flags

Undocumented

QTSPrefsGetActiveConnection

flagsMask

Undocumented

username

Undocumented

password

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPrefsGetActiveConnection

Undocumented

```
OSErr QTSPrefsGetActiveConnection (
    OSType          protocol,
    QTSTransportPref *connectInfo );
```

protocol

A constant (see below) that identifies the connection protocol.

connectInfo

A pointer to a QTSTransportPref (IV–2388) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

protocol Constants

kQTSDirectConnectHTTPProtocol

HTTP protocol; constant value is 'http'.

kQTSDirectConnectRTSPProtocol

RTSP protocol; constant value is 'rtsp'.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPrefsGetNoProxyURLs

Undocumented

```
OSErr QTSPrefsGetNoProxyURLs (
    QTSNoProxyPref  **noProxyHnd1 );
```

noProxyHnd1

A handle to a QTSNoProxyPref (IV-2378) structure.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPrefsSetNoProxyURLs (II-1263).

QTSPrefsSetNoProxyURLs

Undocumented

```
OSErr QTSPrefsSetNoProxyURLs (
    char      *urls,
    UInt32    flags,
    UInt32    seed );
```

urls

A pointer to URL strings.

flags

Undocumented

seed

Undocumented

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

flags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.1.

QTSPresAddSourcer

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPrefsGetNoProxyURLs (II–1263).

QTSPresAddSourcer

Undocumented

```
OSErr QTSPresAddSourcer (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    ComponentInstance   inSourcer,
    SInt32              inFlags );
```

inPresentation

Undocumented

inStream

Undocumented

inSourcer

Undocumented

inFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresExport

Undocumented

```
OSErr QTSPresExport (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    QTSExportParams     *inExportParams );
```

inPresentation

Undocumented

inStream

Undocumented

inExportParams

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresGetActiveSegment

Undocumented

```
OSErr QTSPresGetActiveSegment (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
    TimeValue64        *outStartTime,
    TimeValue64        *outDuration );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

outStartTime

Undocumented

outDuration

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h



QTSPresGetClip

See Also

For the corresponding set function, see QTSPresSetActiveSegment (II–1288).

QTSPresGetClip

Gets the clipping region for a streaming presentation.

```
OSErr QTSPresGetClip (
    QTPresentation    inPresentation,
    QTSSStream        inStream,
    RgnHandle          *outClip );
```

inPresentation

A pointer to a QTPresentationRecord (IV–2379) structure that defines a presentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

outClip

A pointer to a handle to a MacRegion (IV–2303) structure that defines a clipping region.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPresSetClip (II–1289).

QTSPresGetDimensions

Gets the dimensions of a streaming presentation.

```
OSErr QTSPresGetDimensions (
    QTPresentation    inPresentation,
    QTSSStream        inStream,
    Fixed              *outWidth,
    Fixed              *outHeight );
```


`inPresentation`

A pointer to a `QTSPresentationRecord` (IV–2379) structure that defines a presentation.

`inStream`

A pointer to a `QTSSStreamRecord` (IV–2387) structure that defines a stream.

`outWidth`

A pointer to the width in pixels.

`outHeight`

A pointer to the height in pixels.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeStreaming.h`

See Also

For the corresponding set function, see `QTSPresSetDimensions` (II–1290).

QTSPresGetEnable

Determines whether or not a presentation is enabled.

```
OSErr QTSPresGetEnable (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    Boolean              *outEnableMode );
```

`inPresentation`

A pointer to a `QTSPresentationRecord` (IV–2379) structure.

`inStream`

A pointer to a `QTSSStreamRecord` (IV–2387) structure that defines a stream.

`outEnableMode`

A pointer to a `Boolean` that is `TRUE` if the presentation is enabled, `FALSE` otherwise.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

QTSPresGetFlags

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPresSetEnable (II–1290).

QTSPresGetFlags

Gets the flags currently set for a presentation.

```
OSErr QTSPresGetFlags (
    QTSPresentation  inPresentation,
    SInt32            *outFlags );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation.

outFlags

On entry, the address of a variable of type SInt32; on return, this variable is set to the current flags (see below) for the specified presentation.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

outFlags Constants

kQTSAutoModeFlag

Automatic presentation mode.

kQTSDontShowStatusFlag

Don’t show status.

kQTSSendMediaFlag

Send media.

kQTSReceiveMediaFlag

Receive media.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPresSetFlags (II–1291).

QTSPresGetGraphicsMode

Gets the graphics mode and blend color in use for video display by a stream or presentation.

```
OSErr QTSPresGetGraphicsMode (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
    short              *outMode,
    RGBColor            *outOpColor );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation. If you want the graphics mode for a specific stream, pass the value kQTSInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream. If you want the graphics mode for the presentation as a whole, pass the value kQTSAllStreams.

outMode

On entry, a pointer to a short integer; on return, this variable is set to the graphics mode of the specified presentation or stream. See “Graphics Transfer Modes” (IV–2670).

outOpColor

On entry, the address of an RGBColor (IV–2403) structure; on return, this structure is filled in with information about the color used for blending and transparent operations. The stream handler passes this color to QuickDraw as appropriate when you draw in addPin, subPin, blend, or transparent mode.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPresSetGraphicsMode (II–1292).



QTSPresGetGWorld

QTSPresGetGWorld

Gets the graphics port and graphics device in use by a stream or presentation.

```
OSErr QTSPresGetGWorld (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
    CGrafPtr           *outGWorld,
    GDHandle            *outGDHandle );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation. If you want the graphics mode for a specific stream, pass the value kQTInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream. If you want the graphics mode for the presentation as a whole, pass the value kQTAllStreams.

outGWorld

On entry, the address of a variable of type CGrafPtr; on return, this variable is set to a pointer to a CGrafPort (IV–2168) structure that defines the offscreen graphics world, color graphics port, or basic graphics port in use by the specified presentation or stream.

outGDHandle

On entry, the address of a variable of type GDHandle; on return, this variable is set to the handle of a GDevice (IV–2264) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPresSetGWorld (II–1292).

QTSPresGetIndSourcer

Undocumented

```
OSErr QTSPresGetIndSourcer (
```

```
QTSPresentation    inPresentation,
QTSSStream         inStream,
UInt32             inIndex,
ComponentInstance  *outSourcer );
```

inPresentation

Undocumented

inStream

Undocumented

inIndex

Undocumented

outSourcer

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresGetIndStream

Get a stream associated with a presentation, based on its index number.

```
QTSSStream QTSPresGetIndStream (
    QTSPresentation    inPresentation,
    UInt32             inIndex );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inIndex

The index number of the stream.

function result A pointer to a QTSSStreamRecord (IV–2387) structure.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h



QTSPresGetInfo

Gets information about a presentation or stream.

```
OSErr QTSPresGetInfo (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
    OSType              inSelector,
    void                *ioParam );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation. If you want information for a specific stream, pass the value kQTInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream. If you want information for the presentation as a whole, pass the value kQTAllStreams.

inSelector

A constant (see below) that defines the information to be retrieved.

ioParam

A pointer to the retrieved information in the format shown below.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inSelector Constants

kQTSGetURLLink

A URL as a QTSGetURLLinkRecord (IV–2371) structure; constant value is 'gull'.

kQTSTargetBufferDurationInfo

The expected (not necessarily actual) buffer duration in seconds, as a Fixed value; constant value is 'bufr'.

kQTSTargetBufferDurationInfo

The expected (not necessarily actual) buffer duration in seconds, as a Fixed value; constant value is 'bufr'.

kQTSDurationInfo

Duration as a QTSDurationAtom (IV–2369) structure; constant value is 'dura'.

kQTSSourceTrackIDInfo

The source track ID as a UInt32; constant value is 'otid'.

kQTSSourceLayerInfo

The source layer number as a UInt16; constant value is 'olvr'.

kQTSSourceLanguageInfo

The source language code as a UInt16; constant value is 'o1ng'. See “Localization Codes” (IV–2673).

kQTSSourceTrackFlagsInfo

The source track flag set as an SInt32; constant value is 'otfl'.

kQTSSourceDimensionsInfo

The source dimensions as a QTSDimensionParams (IV–2368) structure; constant value is 'odim'.

kQTSSourceVolumesInfo

The source sound volumes as a QTSVolumesParams (IV–2390) structure; constant value is 'ovol'.

kQTSSourceMatrixInfo

The source matrix as a MatrixRecord (IV–2304) structure; constant value is 'omat'.

kQTSSourceClipRectInfo

The source clipping rectangle as a Rect (IV–2399) structure; constant value is 'oclp'.

kQTSSourceGraphicsModeInfo

The source graphics mode as a QTSGraphicsModeParams (IV–2372) structure; constant value is 'ogrm'.

kQTSSourceScaleInfo

The source scaling factors as a Point (IV–2339) structure; constant value is 'oscl'.

kQTSSourceBoundingRectInfo

The source bounding rectangle as a Rect (IV–2399) structure; constant value is 'orct'.

kQTSSourceUserDataInfo

The source’s user data as a UserDataRecord (IV–2496) structure; constant value is 'oudt'.

kQTSSourceInputMapInfo

The source input map as a QT atom container; constant value is 'oimp'.

kQTSStatisticsInfo

Statistics as a QTSStatisticsParams (IV–2384) structure; constant value is 'stat'.

kQTSMInStatusDimensionsInfo

The minimum status dimensions as a QTSDimensionParams (IV–2368) structure; constant value is 'mstd'.



QTSPresGetInfo

kQTSTotalDataRateInfo

The normal status dimensions as a QTSDimensionParams (IV–2368) structure; constant value is 'nstd'.

kQTSTotalDataRateInInfo

The total data rate as a UInt32 added to the existing data rate; constant value is 'drtt'.

kQTSTotalDataRateOutInfo

The total data rate coming in as a UInt32 added to the existing input data rate; constant value is 'drti'.

kQTSTotalDataRateOutInfo

The total data rate going out as a UInt32 added to the existing output data rate; constant value is 'drto'.

kQTSTotalDataRateHistoryInfo

The data rate history as a QTSDataRateHistoryParams (IV–2368) structure; constant value is 'drth'.

kQTSTotalDataRateInfo

The lost data percentage combined with the existing percentage as a QTSTotalDataRateParams (IV–2373) structure; constant value is 'lpct'.

kQTSMediaTypeInfo

The media type as an OSType; constant value is 'mtyp'.

kQTSTotalDataRateInfo

The presentation or stream name as a QTSTotalDataRateParams (IV–2376) structure; constant value is 'name'.

kQTSSampleDescriptionInfo

A handle to the SampleDescription (IV–2414) structure; constant value is 'sdsc'.

kQTSTotalDataRateInfo

Information about whether the stream or presentation can handle the data type as a QTSTotalDataRateParams (IV–2366) structure; constant value is 'chsd'.

kQTSTotalDataRateInfo

Annotations as a QT atom container; constant value is 'meta'.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPresSetInfo (II–1293).

QTSPresGetMatrix

Gets the transformation matrix in use for the graphic display of a stream or presentation.

```
OSErr QTSPresGetMatrix (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    MatrixRecord        *outMatrix );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation. If you want to get the matrix for a specific stream, pass the value kQTInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream. If you want to get the matrix for the presentation as a whole, pass the value kQTSA11Streams.

outMatrix

On entry, the address of a MatrixRecord (IV–2304) structure; on return, this structure is filled with the transformation matrix in use by the stream handler. Note that the matrix passed back is the one last set by QTSPresSetMatrix (II–1295), regardless of any additional matrixes that might have been used.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPresSetMatrix (II–1295).

QTSPresGetNotificationProc

Gets the notification callback of a presentation.

QTSPresGetNumSourcers

```
OSErr QTSPresGetNotificationProc (  
    QTSPresentation      inPresentation,  
    QTSNotificationUPP   *outNotificationProc,  
    void                 **outRefCon );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

outNotificationProc

A pointer to a **Universal Procedure Pointer** that accesses a QTSNotificationProc (III–2126) callback. The callback acts as a back channel from a presentation to its creator. The presentation sends notification of various events, such as a presentation, ending, or acknowledgment of a preroll request.

outRefCon

A handle to a constant to be passed to your QTSNotificationProc.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPresSetNotificationProc (II–1296).

QTSPresGetNumSourcers

Undocumented

```
UInt32 QTSPresGetNumSourcers (  
    QTSPresentation      inPresentation,  
    QTSStream            inStream );
```

inPresentation

Undocumented

inStream

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresGetNumStreams

Undocumented

```
UInt32 QTSPresGetNumStreams (
    QTSPresentation    inPresentation );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

function result *Undocumented*

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresGetPicture

Undocumented

```
OSErr QTSPresGetPicture (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    PicHandle           *outPicture );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

outPicture

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.



QTSPresGetPlayHints

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresGetPlayHints

Undocumented

```
OSErr QTSPresGetPlayHints (
    QTPresentation    inPresentation,
    QTSSStream        inStream,
    SInt32             *outFlags );
```

inPresentation

A pointer to a QTPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

outFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

outFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPresSetPlayHints (II–1296).

QTSPresGetPreferredRate

Undocumented

```
OSErr QTSPresGetPreferredRate (
    QTPresentation    inPresentation,
    Fixed              *outRate );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

outRate

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPresSetPreferredRate (II–1297).

QTSPresGetPresenting

Determines whether presenting is enabled or disabled for a presentation or stream.

```
OSErr QTSPresGetPresenting (
    QTSPresentation  inPresentation,
    QTSSStream       inStream,
    Boolean           *outPresentingMode );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation. If you want to get the presenting state for a specific stream, pass the value kQTSInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream. If you want to get the presenting state for the presentation as a whole, pass the value kQTSAllStreams.

outPresentingMode

A pointer to a Boolean that is TRUE if presenting is enabled, FALSE if it is disabled.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h



QTSPresGetSettings

See Also

For the corresponding set function, see QTSPresSetPresenting (II–1298).

QTSPresGetSettings

Undocumented

```
OSErr QTSPresGetSettings (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    QTAtomContainer     *outSettings,
    SInt32              inFlags );
```

inPresentation

Undocumented

inStream

Undocumented

outSettings

Undocumented

inFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresGetSettingsAsText

Undocumented

```
OSErr QTSPresGetSettingsAsText (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    SInt32              inFlags,
    OSType              inSettingsType,
    Handle              *outText,
    QTSPanelFilterUPP   inPanelFilterProc,
    void                *inPanelFilterProcRefCon );
```

inPresentation

Undocumented

inStream

Undocumented

inFlags

Undocumented

inSettingsType

Undocumented

outText

Undocumented

inPanelFilterProc

Undocumented

inPanelFilterProcRefCon

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresGetTimeBase

Undocumented

```
OSErr QTSPresGetTimeBase (
    QTSPresentation    inPresentation,
    TimeBase            *outTimeBase );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

outTimeBase

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

QTSPresGetTimeScale

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresGetTimeScale

Undocumented

```
OSErr QTSPresGetTimeScale (
    QTSPresentation    inPresentation,
    TimeScale          *outTimeScale );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

outTimeScale

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresGetVolumes

Gets the sound volume levels of a stream or presentation.

```
OSErr QTSPresGetVolumes (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
    short               *outLeftVolume,
    short               *outRightVolume );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation. If you want to get the volumes for a specific stream, pass the value kQTSInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream. If you want to get the volumes for the presentation as a whole, pass the value kQTSAllStreams.

outLeftVolume

On exit, the volume level of the left channel of the stream or presentation. The values returned may range from 0x0000 (silence) to 0x0100 (full volume).

outRightVolume

On exit, the volume level of the right channel of the stream or presentation. The values returned may range from 0x0000 (silence) to 0x0100 (full volume).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding set function, see QTSPresSetVolumes (II–1301).

QTSPresHasCharacteristic

Undocumented

```
OSErr QTSPresHasCharacteristic (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    OSType              inCharacteristic,
    Boolean             *outHasIt );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

inCharacteristic

Undocumented

outHasIt

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h



QTSPresIdle

QTSPresIdle

Undocumented

```
void QTSPresIdle (
    QTSPresentation      inPresentation,
    QTSPresIdleParams    *ioParams );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

ioParams

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresInvalidateRegion

Undocumented

```
OSErr QTSPresInvalidateRegion (
    QTSPresentation      inPresentation,
    RgnHandle             inRegion );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inRegion

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresNewStream

Undocumented

```
OSErr QTSPresNewStream (
    QTSPresentation    inPresentation,
    OSType              inDataType,
    const void          *inData,
    UInt32              inDataLength,
    SInt32              inFlags,
    QTSSStream          *outStream );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inDataType

Undocumented

inData

Undocumented

inDataLength

Undocumented

inFlags

Undocumented

outStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresPreroll

Undocumented

```
OSErr QTSPresPreroll (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    UInt32              inTimeValue,
    Fixed               inRate,
    SInt32              inFlags );
```

QTSPresPreroll64

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

inTimeValue

Undocumented

inRate

Undocumented

inFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresPreroll64

Undocumented

```
OSErr QTSPresPreroll64 (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    const TimeValue64   *inPrerollTime,
    Fixed               inRate,
    SInt32               inFlags );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

inPrerollTime

Undocumented

inRate

Undocumented

inFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresPreview

Undocumented

```
OSErr QTSPresPreview (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    const TimeValue64  *inTimeValue,
    Fixed               inRate,
    SInt32              inFlags );
```

inPresentation

Undocumented

inStream

Undocumented

inTimeValue

Undocumented

inRate

Undocumented

inFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.



QTSPresRemoveSourcer

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresRemoveSourcer

Undocumented

```
OSErr QTSPresRemoveSourcer (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    ComponentInstance   inSourcer,
    SInt32              inFlags );
```

inPresentation

Undocumented

inStream

Undocumented

inSourcer

Undocumented

inFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresSetActiveSegment

Undocumented

```
OSErr QTSPresSetActiveSegment (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    const TimeValue64   *inStartTime,
    const TimeValue64   *inDuration );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

inStartTime

Undocumented

inDuration

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetActiveSegment (II–1265).

QTSPresSetClip

Undocumented

```
OSErr QTSPresSetClip (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    RgnHandle           inClip );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

inClip

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetClip (II–1266).



QTSPresSetDimensions

QTSPresSetDimensions

Undocumented

```
OSErr QTSPresSetDimensions (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
    Fixed              inWidth,
    Fixed              inHeight );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

inWidth

Undocumented

inHeight

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetDimensions (II–1266).

QTSPresSetEnable

Undocumented

```
OSErr QTSPresSetEnable (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
    Boolean             inEnableMode );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

inEnableMode

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetEnable (II–1267).

QTSPresSetFlags

Undocumented

```
OSErr QTSPresSetFlags (
    QTSPresentation    inPresentation,
    SInt32              inFlags,
    SInt32              inFlagsMask );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inFlags

Undocumented

inFlagsMask

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetFlags (II–1268).



QTSPresSetGraphicsMode

QTSPresSetGraphicsMode

Sets the graphics transfer mode for a streaming presentation.

```
OSErr QTSPresSetGraphicsMode (
    QTPresentation    inPresentation,
    QTSSStream        inStream,
    short              inMode,
    const RGBColor     *inOpColor );
```

inPresentation

A pointer to a QTPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

inMode

A short integer; see “Graphics Transfer Modes” (IV–2670).

inOpColor

A pointer to an RGBColor (IV–2403) structure. This is the blend value for blends and the transparent color for transparent operations. The toolbox supplies this value to QuickDraw when you draw in addPin, subPin, blend, transparent, or graphicsModeStraightAlphaBlend mode.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetGraphicsMode (II–1269).

QTSPresSetGWorld

Undocumented

```
OSErr QTSPresSetGWorld (
    QTPresentation    inPresentation,
    QTSSStream        inStream,
    CGrafPtr           inGWorld,
    GDHandle           inGDHandle );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream.

inGWorld

Undocumented

inGDHandle

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetGWorld (II–1270).

QTSPresSetInfo

Sets information for a presentation or stream.

```
OSErr QTSPresSetInfo (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    OSType              inSelector,
    void                *ioParam );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation. If you want to set information for a specific stream, pass the value kQTSInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream. If you want to set information for the presentation as a whole, pass the value kQTSAllStreams.

inSelector

A constant (see below) that defines the type of information to be set.

QTSPresSetInfo

ioParam

A pointer to the information to be set in the format shown below.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inSelector Constants

`kQTSGetURLLink`

A URL as a `QTSGetURLLinkRecord` (IV–2371) structure; constant value is `'gull'`.

`kQTSTargetBufferDurationInfo`

The expected (not necessarily actual) buffer duration in seconds, as a Fixed value; constant value is `'bufrr'`.

`kQTSDurationInfo`

Duration as a `QTSDurationAtom` (IV–2369) structure; constant value is `'dura'`.

`kQTSSourceTrackIDInfo`

The source track ID as a `UInt32`; constant value is `'otid'`.

`kQTSSourceLayerInfo`

The source layer number as a `UInt16`; constant value is `'olrr'`.

`kQTSSourceLanguageInfo`

The source language code as a `UInt16`; constant value is `'olng'`. See “Localization Codes” (IV–2673).

`kQTSSourceTrackFlagsInfo`

The source track flag set as an `SInt32`; constant value is `'otfl'`.

`kQTSSourceDimensionsInfo`

The source dimensions as a `QTSDimensionParams` (IV–2368) structure; constant value is `'odim'`.

`kQTSSourceVolumesInfo`

The source sound volumes as a `QTSVolumesParams` (IV–2390) structure; constant value is `'ovol'`.

`kQTSSourceMatrixInfo`

The source matrix as a `MatrixRecord` (IV–2304) structure; constant value is `'omat'`.

`kQTSSourceClipRectInfo`

The source clipping rectangle as a `Rect` (IV–2399) structure; constant value is `'oclp'`.

`kQTSSourceGraphicsModeInfo`

The source graphics mode as a `QTSGraphicsModeParams` (IV–2372) structure; constant value is `'ogrm'`.

kQTSSourceScaleInfo

The source scaling factors as a Point (IV–2339) structure; constant value is 'oscl'.

kQTSSourceBoundingRectInfo

The source bounding rectangle as a Rect (IV–2399) structure; constant value is 'orct'.

kQTSSourceUserDataInfo

The source's user data as a UserDataRecord (IV–2496) structure; constant value is 'oudt'.

kQTSSourceInputMapInfo

The source input map as a QT atom container; constant value is 'oimp'.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetInfo (II–1272).

QTSPresSetMatrix

Sets the transformation matrix to be used by the graphic display of a stream or presentation.

```
OSErr QTSPresSetMatrix (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    const MatrixRecord  *inMatrix );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation. If you want to set the matrix for a specific stream, pass the value kQTSTInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream. If you want to set the matrix for the presentation as a whole, pass the value kQTSA11Streams.

inMatrix

A pointer to a MatrixRecord (IV–2304) structure.

QTSPresSetNotificationProc

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetMatrix (II–1275).

QTSPresSetNotificationProc

Sets the notification callback for a presentation.

```
OSErr QTSPresSetNotificationProc (  
    QTPresentation      inPresentation,  
    QTSNotificationUPP  inNotificationProc,  
    void                *inRefCon );
```

inPresentation

A pointer to a QTPresentationRecord (IV–2379) structure.

inNotificationProc

A **Universal Procedure Pointer** that accesses a QTSNotificationProc (III–2126) callback. The callback acts as a back channel from a presentation to its creator. The presentation sends notification of various events, such as a presentation, ending, or acknowledgment of a preroll request.

inRefCon

A pointer to data to be passed to your QTSNotificationProc.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetNotificationProc (II–1275).

QTSPresSetPlayHints

Undocumented

```
OSErr QTSPresSetPlayHints (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
    SInt32              inFlags,
    SInt32              inFlagsMask );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation. If you want to set the play hints for a specific stream, pass the value kQTSInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream. If you want to set the play hints for the presentation as a whole, pass the value kQTSAllStreams.

inFlags

Undocumented

inFlagsMask

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetPlayHints (II–1278).

QTSPresSetPreferredRate

Undocumented

```
OSErr QTSPresSetPreferredRate (
    QTSPresentation    inPresentation,
    Fixed               inRate,
    SInt32              inFlags );
```

QTSPresSetPresenting

`inPresentation`

A pointer to a `QTSPresentationRecord` (IV–2379) structure.

`inRate`

Undocumented

`inFlags`

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeStreaming.h`

See Also

For the corresponding get function, see `QTSPresGetPreferredRate` (II–1278).

QTSPresSetPresenting

Enables or disables presentation of a stream to the user.

```
OSErr QTSPresSetPresenting (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    Boolean             inPresentingMode );
```

`inPresentation`

A pointer to a `QTSPresentationRecord` (IV–2379) structure that defines a presentation. If you want to enable or disable the presentation for a specific stream, pass the value `kQTSInvalidPresentation`.

`inStream`

A pointer to a `QTSSStreamRecord` (IV–2387) structure that defines a stream. If you want to enable or disable the presentation as a whole, pass the value `kQTSAllStreams`.

`inPresentingMode`

Pass `TRUE` to enable the presentation, `FALSE` to disable it.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetPresenting (II–1279).

QTSPresSetSettings

Undocumented

```
OSErr QTSPresSetSettings (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
    QTAtomSpecPtr      inSettings,
    Sint32             inFlags );
```

inPresentation

Undocumented

inStream

Undocumented

inSettings

Undocumented

inFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresSettingsDialog

Undocumented

```
OSErr QTSPresSettingsDialog (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
```



QTSPresSettingsDialogWithFilters

```
SInt32          inFlags,  
QTSMoAlFilterUPP inFilterProc,  
void            *inFilterProcRefCon );  
  
inPresentation  
    Undocumented  
  
inStream  
    Undocumented  
  
inFlags  
    Undocumented  
  
inFilterProc  
    Undocumented  
  
inFilterProcRefCon  
    Undocumented  
  
function result    See “Error Codes” (IV–2663). Returns noErr if there is no error.
```

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresSettingsDialogWithFilters

Undocumented

```
OSErr QTSPresSettingsDialogWithFilters (  
    QTSPPresentation    inPresentation,  
    QTSStream           inStream,  
    SInt32              inFlags,  
    QTSMoAlFilterUPP    inFilterProc,  
    void                *inFilterProcRefCon,  
    QTSPPanelFilterUPP  inPanelFilterProc,  
    void                *inPanelFilterProcRefCon );  
  
inPresentation  
    Undocumented  
  
inStream  
    Undocumented  
  
inFlags  
    Undocumented
```

inFilterProc

Undocumented

inFilterProcRefCon

Undocumented

inPanelFilterProc

Undocumented

inPanelFilterProcRefCon

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresSetVolumes

Sets the sound volume levels of a stream or presentation.

```
OSErr QTSPresSetVolumes (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    short               inLeftVolume,
    short               inRightVolume );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation. If you want to set the volume of a specific stream, pass the value kQTSInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream. If you want to set the volume of the presentation as a whole, pass the value kQTSA11Streams.

inLeftVolume

The volume level to be set for the left channel of the stream or presentation. The values may range from 0x0000 (silence) to 0x0100 (full volume).

inRightVolume

The volume level to be set for the right channel of the stream or presentation. The values may range from 0x0000 (silence) to 0x0100 (full volume).

QTSPresSkipTo

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSPresGetVolumes (II–1282).

QTSPresSkipTo

Requests that a presentation skip to a given point, specified by a time value.

```
OSErr QTSPresSkipTo (
    QTPresentation    inPresentation,
    UInt32             inTimeValue );
```

inPresentation

A pointer to a QTPresentationRecord (IV–2379) structure.

inTimeValue

The time value to skip to, expressed in the time scale of the presentation.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresSkipTo64

Requests that a streaming presentation skip to a given point, specified by a 64-bit time value.

```
OSErr QTSPresSkipTo64 (
    QTPresentation    inPresentation,
    const TimeValue64 *inTimeValue );
```

inPresentation

A pointer to a QTPresentationRecord (IV–2379) structure.

inTimeValue

A pointer to a signed 64-bit integer that contains the time value to skip to, expressed in the time scale of the presentation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `QuickTimeStreaming.h`

QTSPresStart

Starts a streaming presentation or a stream.

```
OSErr QTSPresStart (
    QTSPresentation    inPresentation,
    QTSSStream         inStream,
    SInt32             inFlags );
```

inPresentation

A pointer to a `QTSPresentationRecord` (IV–2379) structure that defines a presentation. If you want to start a specific stream, pass the value `kQTSInvalidPresentation`.

inStream

A pointer to a `QTSSStreamRecord` (IV–2387) structure that defines a stream. If you want to start the presentation as a whole, pass the value `kQTSA11Streams`.

inFlags

Flags (see below) that govern the starting of the presentation or stream.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inFlags Constants

`kQTSSyncControlFlag`

Undocumented

`kQTSTimeBaseRunningControlFlag`

Undocumented

Discussion

If `QTSPresPreroll` (II–1285) has not been called, QuickTime must set up the streams and do everything that would have been done in preroll. If the presentation has already been prerolled, it should be ready to start immediately.

QTSPresStop

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresStop

Stops a streaming presentation or stream.

```
OSErr QTSPresStop (
    QTSPresentation    inPresentation,
    QTSSStream          inStream,
    SInt32              inFlags );
```

inPresentation

A pointer to a QTSPresentationRecord (IV–2379) structure that defines a presentation. If you want to stop a specific stream, pass the value kQTSInvalidPresentation.

inStream

A pointer to a QTSSStreamRecord (IV–2387) structure that defines a stream. If you want to stop the presentation as a whole, pass the value kQTSAllStreams. All audio and video output will cease.

inFlags

Flags that govern the stopping of the presentation or stream. No flags are currently defined.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSReleaseMemPtr

Disposes of a pointer to a streaming buffer that will be recirculated.

```
void QTSReleaseMemPtr (
    QTSMemPtr    inMemPtr,
    SInt32        inFlags );
```

inMemPtr

A pointer to an opaque structure.

inFlags

Undocumented

inFlags Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTSSetNetworkAppName

Sets the name of a streaming network application.

```
OSErr QTSSetNetworkAppName (
    const char    *inAppName,
    SInt32        inFlags );
```

inAppName

A pointer to a string containing the application's name.

inFlags

A flag (see below) that determines whether the name is a full pathname.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inFlags Constants

kQTSNetworkAppNameIsFullNameFlag

The application name is a full pathname.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

For the corresponding get function, see QTSGetNetworkAppName (II–1237).



QTSSourcerGetEnable

QTSSourcerGetEnable

Undocumented

```
ComponentResult QTSSourcerGetEnable (
    QTSSourcer    inSourcer,
    Boolean       *outEnableMode,
    SInt32        inFlags );
```

inSourcer

Undocumented

outEnableMode

Undocumented

inFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

QTSSourcerGetInfo

Undocumented

```
ComponentResult QTSSourcerGetInfo (
    QTSSourcer    inSourcer,
    OSType        inSelector,
    void          *ioParams );
```

inSourcer

Undocumented

inSelector

Undocumented

ioParams

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

QTSSourcerGetTimeScale

Undocumented

```
ComponentResult QTSSourcerGetTimeScale (
    QTSSourcer    inSourcer,
    TimeScale      *outTimeScale );
```

inSourcer

Undocumented

outTimeScale

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

QTSSourcerIdle

Undocumented

```
ComponentResult QTSSourcerIdle (
    QTSSourcer    inSourcer,
    const TimeValue64 *inTime,
    SInt32         inFlags,
    SInt32         *outFlags );
```

inSourcer

Undocumented

inTime

Undocumented

QTSSourcerInitialize

inFlags

Undocumented

outFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

QTSSourcerInitialize

Undocumented

```
ComponentResult QTSSourcerInitialize (  
    QTSSourcer          inSourcer,  
    const QTSSourcerInitParams *inInitParams );
```

inSourcer

Undocumented

inInitParams

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

QTSSourcerSetEnable

Undocumented

```
ComponentResult QTSSourcerSetEnable (  
    QTSSourcer    inSourcer,  
    Boolean        inEnableMode,  
    SInt32         inFlags );
```

inSourcer

Undocumented

inEnableMode

Undocumented

inFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

QTSSourcerSetInfo

Undocumented

```
ComponentResult QTSSourcerSetInfo (
    QTSSourcer    inSourcer,
    OSType        inSelector,
    void          *ioParams );
```

inSourcer

Undocumented

inSelector

Undocumented

ioParams

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

QTSSourcerSetTimeScale

Undocumented



QTStatHelperGetNumStats

```
ComponentResult QTSSourcerSetTimeScale (  
    QTSSourcer    inSourcer,  
    TimeScale     inTimeScale );
```

inSourcer

Undocumented

inTimeScale

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

QTStatHelperGetNumStats

Gets the number of statistics that a statistic helper is reporting.

```
UInt32 QTStatHelperGetNumStats (  
    QTStatHelper    inStatHelper );
```

inStatHelper

A pointer to a QTStatHelperRecord (IV–2384) structure that defines the component instance of a statistics helper.

function result The number of statistics.

Discussion

You can also find the number of statistics that a statistics helper is reporting by calling QTStatHelperResetIter (II–1312), then calling QTStatHelperNext (II–1311) iteratively until it returns FALSE and counting the iterations.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTStatHelperGetStats

Tells a statistics helper to update its statistics.

```
OSErr QTStatHelperGetStats (
    QTStatHelper    inStatHelper );
```

inStatHelper

A pointer to a QTStatHelperRecord (IV–2384) structure that defines the component instance of a statistics helper.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Statistics helpers update their statistics only when this function is called. You should call it at least once before calling QTStatHelperNext (II–1311), to ensure that the information returned is valid and current. The normal sequence is to call this function, then call QTStatHelperResetIter (II–1312), then make a series of calls to QTStatHelperNext (II–1311).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTStatHelperNext

Gets the next statistic from a statistic helper.

```
Boolean QTStatHelperNext (
    QTStatHelper    inStatHelper,
    QTStatHelperNextParams    *ioParams );
```

inStatHelper

A pointer to a QTStatHelperRecord (IV–2384) structure that defines the component instance of a statistics helper.

ioParams

On entry, a pointer to a QTStatHelperNextParams (IV–2382) structure; on return, this structure is filled in with information about the next statistic from the specified statistic helper.

function result FALSE if the last statistic has been returned, TRUE otherwise.

Discussion

You need to call this function once to retrieve each statistic. The normal sequence is to call QTStatHelperGetStats (II–1310), then call QTStatHelperResetIter (II–1312), then make a series of calls to this function until it returns FALSE.

QTStatHelperResetIter

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

QTStatHelperResetIter

Reset the iteration counter of a statistics helper, so the next call to QTStatHelperNext (II–1311) returns the first statistic.

```
OSErr QTStatHelperResetIter (
    QTStatHelper    inStatHelper );
```

inStatHelper

A pointer to a QTStatHelperRecord (IV–2384) structure that defines the component instance of a statistics helper.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

See QTStatHelperNext (II–1311).

QTStreamBufferDataInfo

Undocumented

```
void QTStreamBufferDataInfo (
    QTStreamBuffer    *inStreamBuffer,
    unsigned char      **outDataStart,
    UInt32             *outDataMaxLength );
```

inStreamBuffer

Undocumented

outDataStart

Undocumented

outDataMaxLength

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeStreaming.h

QTStandardParameterDialogDoAction

Lets you change some of the default behaviors of the standard parameter dialog box.

```
OSErr QTStandardParameterDialogDoAction (
    QTParameterDialog    createdDialog,
    long                 action,
    void                  *params );
```

createdDialog

The reference to the dialog box created by calling
QTCreateStandardParameterDialog (II–1161).

action

Determines which of the actions (see below) supported by this function will be performed.

params

Optional parameters to the action. The type passed in this parameter depends on the value of the action parameter.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

action Constants

pdActionSetAppleMenu

Use this selector to make the Apple menu available while the standard parameter dialog box is active. Pass the MenuHandle of the Apple menu in the params parameter.

QTStandardParameterDialogDoAction

pdActionSetEditMenu

Use this action to make your application's Edit menu available while the standard parameters dialog box is active. Pass the `MenuHandle` of the Edit menu in the `params` parameter.

pdActionSetPreviewPicture

Use this action to change the images that are used in the **preview** window of the standard parameters dialog box. QuickTime provides default images but you may wish, for example, to use thumbnail images taken from your application instead. The `params` parameter contains a `QTParamPreviewPtr` referencing the image to use.

pdActionSetDialogTitle

Sets the title of the standard parameters dialog box. The `params` parameter contains a Pascal `Str255` string.

pdActionGetSubPanelMenu

Returns a list of the sub-panels used by the standard parameters dialog box. If there are more parameter controls than can fit into the available area of the dialog box, the parameters are automatically split into sub-panels (usually by segmenting the set of parameters by group). The pop-up menu used to switch between the panels is returned as a `MenuHandle` in the `params` parameter.

pdActionActivateSubPanel

Sets the current sub-panel. The `params` parameter contains a long integer that is the number of the panel to be displayed.

pdActionConductStopAlert

Displays a Stop alert. The `params` parameter contains a `StringPtr` that is displayed in the alert. This feature can be used in conjunction with `ImageCodecValidateParameters` (II-745) to inform the user when invalid parameter values have been entered.

Discussion

This function allows you to change some of the default behaviors of a standard parameter dialog box you create using the `QTCreateStandardParameterDialog` (II-1161) function. To choose which of the available customizations to perform, pass an action selector value in the `action` parameter and, optionally, a single parameter in `params`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "High-Level Effects Functions" (V-2897)

QTSwapAtoms

Swaps the contents of two atoms in an atom container.

```
OSErr QTSwapAtoms (
    QAtomContainer    container,
    QAtom             atom1,
    QAtom             atom2 );
```

container

The atom container for this operation.

atom1

Specifies an atom to be swapped with the atom specified by atom2.

atom2

Specifies an atom to be swapped with the atom specified by atom1.

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

After swapping, the ID and index of each atom remains the same. The two atoms specified must be of the same type. Either atom may be a leaf atom or a container atom.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Copying Existing Atoms” (V-2767)

Related Java Methods

`quicktime.std.movies.AtomContainer.swapAtoms()`

QTTextToNativeText

Undocumented

```
OSErr QTTextToNativeText (
    Handle    theText,
    long      encoding,
    long      flags );
```

QTUnlockContainer

theText

Undocumented

encoding

Undocumented

flags

Flags (see below) that define the text atom type.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

flags Constants

`kITextAtomType`

International text atom; value is 'itxt'.

`kITextStringAtomType`

String atom; value is 'text'.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

QTUnlockContainer

Unlocks an atom container in memory.

```
OSErr QTUnlockContainer (
    QTAtomContainer  container );
```

container

The atom container to be unlocked.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You should call this function to unlock an atom container when you have finished accessing a leaf atom’s data. You may make nested pairs of calls to `QTLockContainer` (II–1187) and this function; you don’t need to check the current state of the container first.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Retrieving Atoms and Atom Data” (V–2768)

QTUnregisterAccessKey

Removes a previously registered access key.

```
OSErr QTUnregisterAccessKey (
    Str255    accessKeyType,
    long      flags,
    Handle     accessKey );
```

`accessKeyType`

The access key type of the key to be removed.

`flags`

Flags (see below) that specify the operation of this function. To remove a system access key, set the `kAccessKeySystemFlag` flag. To remove an application access key, set this parameter to 0.

`accessKey`

The key to be removed.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

flags Constant

`kAccessKeySystemFlag`

Set to 1 to remove a system access key.

Discussion

Most access keys are strings. A string stored in the `accessKey` handle does not include a trailing zero or a leading length byte.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Registering and Unregistering Access Keys” (V–2771)

QTUpdateGWorld

Changes the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world with a non-Macintosh pixel format.

```
GWorldFlags QTUpdateGWorld (  
    GWorldPtr      *offscreenGWorld,  
    OSType         PixelFormat,  
    const Rect     *boundsRect,  
    CTabHandle     cTable,  
    GDHandle       aGDevice,  
    GWorldFlags    flags );
```

offscreenGWorld

On input, a pointer to an existing offscreen graphics world; upon completion, the pointer to the updated offscreen graphics world.

PixelFormat

The updated graphics world's pixel format; see "Pixel Formats" (IV-2686).

boundsRect

A pointer to the boundary rectangle and port rectangle for the updated offscreen pixel map. This becomes the boundary rectangle for the GDevice structure, if this function creates one. If you specify 0 in the PixelFormat parameter, the function interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. It then uses the pixel format, color table, and GDevice structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle. If the rectangle you specify in this parameter differs from, but has the same size as, the previous boundary rectangle, the function realigns the pixel image to the screen for optimum performance for CopyBits. Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

cTable

A handle to a ColorTable (IV-2210) structure for the updated graphics world. If you pass NIL in this parameter, the function uses the default color table for the pixel format that you specify in the PixelFormat parameter. If you set the PixelFormat parameter to 0, the function ignores the cTable parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the boundsRect parameter. If the color table that you specify in this parameter is different from the previous color table, or if the

color table associated with the `GDevice` structure that you specify in the `aGDevice` parameter is different, the function maps the pixel values in the offscreen pixel map to the new color table. If you use this function on a computer that supports only basic QuickDraw, you may specify only `NIL` in this parameter.

`aGDevice`

A handle to a `GDevice` (IV-2264) structure that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case the function attaches this structure to the new offscreen graphics world. If you set the `PixelFormat` parameter to 0, or if you do not set the `noNewDevice` flag, the function ignores this parameter, so you should set it to `NIL`. If you set the `PixelFormat` parameter to 0, the function uses the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NIL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` structures for offscreen graphics worlds.

`flags`

Constants (see below) that identify options available to your application. You can set a combination of these flags. If you don't wish to use any of them, pass 0 in this parameter. In this case the default behavior is to create an offscreen graphics world where the base address for the offscreen pixel image is un purgeable, the graphics world uses an existing `GDevice` structure (if you pass 0 in the `depth` parameter) or creates a new `GDevice` structure, it uses memory in your application heap, and it allows graphics accelerators to cache the offscreen pixel image.

function result A constant (see below) that reports on the operation of this function.

flags Constants

`pixPurge`

Make base address for the offscreen pixel image purgeable.

`noNewDevice`

Do not create an offscreen `GDevice` structure.

`useTempMem`

Create a base address for the offscreen pixel image in temporary memory.

`keepLocal`

Keep the offscreen pixel image in main memory where it cannot be cached to a graphics accelerator card.

Function Result Constants

`gwFlagErr`

Function unsuccessful; the offscreen graphics world has been left unchanged.

`mapPix`

Colors were remapped to a new color table.

`newDepth`

The pixel values in the offscreen pixel image were translated to those for a different pixel depth.

`alignPix`

The offscreen image was realigned to the window.

`newRowBytes`

The value of the `rowBytes` field in the `PixelFormat` structure for the offscreen graphics world was changed.

`reallocPix`

Memory for the offscreen pixel image had to be reallocated; your application should then reconstruct the pixel image, or draw directly in a window instead of preparing the image in an offscreen graphics world.

`clipPix`

The pixel image was clipped. If the rectangle you specified in the `boundsRect` parameter is smaller than the previous boundary rectangle the pixel image was clipped along the bottom and right edges. If the rectangle you specified was bigger than the previous boundary rectangle, the bottom and right edges of the pixel image are undefined.

`stretchPix`

The offscreen image was stretched or shrunk. If the rectangle you specified in the `boundsRect` parameter was smaller than the previous boundary rectangle, the pixel image is reduced to the new size. If the rectangle was bigger than the previous boundary rectangle, the pixel image was stretched to the new size.

`ditherPix`

The offscreen pixel image was **dithered**.

Discussion

If the Memory Manager purged the base address for the offscreen pixel image, this function reallocates the memory but the pixel image is lost. You must reconstruct it.

Special Considerations

This function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

QTVideoOutputBegin

Obtains exclusive access to the video hardware controlled by a video output component.

```
ComponentResult QTVideoOutputBegin (
    QTVideoOutputComponent  vo );
```

vo

The instance of a video output component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. If this function returns the `videoOutputInUseErr` result code that indicates that the video hardware is currently in use, your software can get the name of the application or other software that is using the hardware by calling `QTVideoOutputGetCurrentClientName` (II–1325). You can then display an alert to the user that says that the video hardware is in use and specifies the name of the software using the video hardware.

Discussion

When your software calls this function, the video output component acquires exclusive access to the video hardware controlled by the specified video output component or returns the `videoOutputInUseErr` result code if the video hardware is currently in use. If the video hardware is available, the video output component also enables the display mode last set with `QTVideoOutputSetDisplayMode` (II–1332) and enables the video settings, if any, that were most recently specified by the user in a custom video configuration dialog box. If the video output component supports `QTVideoOutputCustomConfigureDisplay` (II–1322), your software can call the function to display a custom video configuration dialog box. When your software no longer needs the video output component, release it by calling `QTVideoOutputEnd` (II–1322).

Special Considerations

If your software needs to change the display mode, it must change it before calling this function. It cannot change the display mode between calls to this function and to `QTVideoOutputEnd` (II–1322).

QTVideoOutputCustomConfigureDisplay

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Video Output” (V–2893)

QTVideoOutputCustomConfigureDisplay

Displays a custom video configuration dialog box, which can include settings that are specific to the video device controlled by the video output component.

```
ComponentResult QTVideoOutputCustomConfigureDisplay (
    QTVideoOutputComponent    vo,
    ModalFilterUPP             filter );
```

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

filter

A `ModalFilterProc` (III–2098) callback for the video output component to use for the dialog box. The filter allows the software to process events while the dialog box is displayed.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your software can determine if a video output component supports this function by calling `ComponentFunctionImplemented` (I–111) for the component with the routine selector `kQTVideoOutputCustomConfigureDisplaySelect`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Video Output” (V–2893)

QTVideoOutputEnd

Releases access to the video hardware controlled by a video output component.


```
ComponentResult QTVideoOutputEnd (
    QTVideoOutputComponent    vo );
```

vo

The instance of a video output component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your software should release access to a video output component as soon as it is done using the video hardware controlled by the component. If you close the instance of a video output component that currently has exclusive access to video hardware, the video output component automatically calls this function to release the hardware.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Video Output” (V–2893)

QTVideoOutputGetClientName

Obtains the name of the application or other software that is registered with an instance of a video output component.

```
ComponentResult QTVideoOutputGetClientName (
    QTVideoOutputComponent    vo,
    Str255                     str );
```

vo

The instance of a video output component for the request. Your software obtains this reference when it calls `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

str

The name of the application or other software that is registered with the component instance.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

QTVideoOutputGetClock

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Registering the Name of Video Output Software” (V–2892)

See Also

For the corresponding set function, see `QTVideoOutputSetClientName` (II–1332).

QTVideoOutputGetClock

Returns a pointer to the clock component associated with the video output component.

```
ComponentResult QTVideoOutputGetClock (
    QTVideoOutputComponent    vo,
    ComponentInstance          *clock );
```

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

clock

A pointer to the clock component associated with the video output component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your software can use the clock component returned by this function to synchronize video and sound for a movie to the rate of the display. To associate the instance of the clock component with a movie, call `SetMovieMasterClock` (III–1620). Because a change to the display mode could affect a clock component, your software should call this function only between calls to `QTVideoOutputBegin` (II–1321) and `QTVideoOutputEnd` (II–1322), when it is not possible to change the display mode.

Special Considerations

When your software calls `QTVideoOutputEnd` (II–1322), the video output component disposes of the instance of the clock component returned by this function. Because of this, software that uses the clock to control a movie must reset the clock for the movie to the default clock, by calling `SetMovieMasterClock` (III–1620) with `NIL` as the value of the clock component, before calling `QTVideoOutputEnd`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Finding Components Associated With a Video Output” (V–2894)

QTVideoOutputGetCurrentClientName

Returns the name of the software, if any, that has exclusive access to the video hardware controlled by a video output component.

```
ComponentResult QTVideoOutputGetCurrentClientName (
    QTVideoOutputComponent  vo,
    Str255                  str );
```

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

str

The name of the software that has exclusive access to the video hardware controlled by a video output component, or a zero-length string if no software currently has access.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If video hardware is unavailable because other software is using it, your software can inform users by getting the name of the software with this function and displaying the name in an alert box.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Registering the Name of Video Output Software” (V–2892)

QTVideoOutputGetDisplayMode

Returns the current display mode for a video output component.

QTVideoOutputGetDisplayModeList

```
ComponentResult QTVideoOutputGetDisplayMode (
    QTVideoOutputComponent    vo,
    long                      *displayModeID );
```

vo

The instance of a video output component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

displayModeID

A pointer to the ID of the current display mode, or 0 if no display mode has been selected. The ID specifies a QT atom of type `kQTVODisplayModeItem` in the QT atom container returned by `QTVideoOutputGetDisplayModeList` (II–1326).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. If this function returns an atom ID of 0, it indicates that no display mode has been selected.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling the Video Output Display Mode” (V–2892)

See Also

For the corresponding set function, see `QTVideoOutputSetDisplayMode` (II–1332).

QTVideoOutputGetDisplayModeList

Returns a list of the display modes supported by a video output component.

```
ComponentResult QTVideoOutputGetDisplayModeList (
    QTVideoOutputComponent    vo,
    QTAtomContainer           *outputs );
```

vo

The instance of a video output component. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

outputs

A pointer to the QT atom container that lists the video modes supported by this component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

After your software calls this function, it must dispose of the QT atom container returned by the function by calling `QTDisposeAtomContainer` (II–1164).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling the Video Output Display Mode” (V–2892)

QTVideoOutputGetGWorld

Returns a pointer to the graphics world used by a video output component.

```
ComponentResult QTVideoOutputGetGWorld (
    QTVideoOutputComponent    vo,
    GWorldPtr                  *gw );
```

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

gw

A pointer to the graphics world used by the video output component to display images.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If the pixel format of the graphics world is 1, 2, 4, 8, 16, or 32, your software can use either `QuickDraw` or `QuickTime` to draw graphics to it. If the graphics world has any other pixel format, your software must use `QuickTime` functions `draw` to it. Your software can pass the pointer returned by this function to the `SetMovieGWorld` (III–1619), `DecompressSequenceBegin` (I–238), `DecompressSequenceBeginS` (I–239), `DecompressImage` (I–235), and `FDecompressImage` (I–355) functions.

Your software can call `QTVideoOutputGetGWorld` only between calls to `QTVideoOutputBegin` (II–1321) and `QTVideoOutputEnd` (II–1322). When your software calls `QTVideoOutputEnd`, the video output component automatically disposes of the graphics world. If your software needs to use the graphics world after calling `QTVideoOutputEnd`, it must call this function again after the next time it calls `QTVideoOutputBegin`.

QTVideoOutputGetGWorldParameters

Special Considerations

Your software must not dispose of the graphics world used by a video output component. The video output component automatically disposes of the graphics world when your software calls `QTVideoOutputEnd` (II–1322).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Video Output” (V–2893)

QTVideoOutputGetGWorldParameters

Called by the base video output component as part of its implementation of `QTVideoOutputGetGWorld` (II–1327).

```
ComponentResult QTVideoOutputGetGWorldParameters (
    QTVideoOutputComponent    vo,
    Ptr                        *baseAddr,
    long                       *rowBytes,
    CTabHandle                 *colorTable );
```

vo

An instance of your video output component.

baseAddr

The address at which to display pixels. If your video output component does not display pixels, return 0 for this parameter.

rowBytes

The width of each scan line in bytes. If your video output component does not display pixels, return the width of the current display mode.

colorTable

The `ColorTable` (IV–2210) structure to be used. If your video output component does not use a color table, return `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is not called by applications or other client software.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Video Output” (V–2893)

QTVideoOutputGetIndImageDecompressor

Undocumented

```
ComponentResult QTVideoOutputGetIndImageDecompressor (
    QTVideoOutputComponent  vo,
    long                    index,
    Component                *codec );
```

vo

Undocumented

index

Undocumented

codec

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

QTVideoOutputGetIndSoundOutput

Determines which sound output components are associated with the video output component.

```
ComponentResult QTVideoOutputGetIndSoundOutput (
    QTVideoOutputComponent  vo,
    long                    index,
    Component                *outputComponent );
```

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

QTVideoOutputRestoreState

index

Specifies which of the sound output components to return. The index of the first component is 1.

outputComponent

A pointer to a sound output component associated with the video output component that is specified by the *index* parameter.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Discussion

Your software can display sound output components returned by this function in a dialog box and let the user choose which outputs to use for movie playback.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: *QuickTimeComponents.h*

Programming summary: “Finding Components Associated With a Video Output” (V–2894)

QTVideoOutputRestoreState

Restores the previously saved state of a video output component.

```
ComponentResult QTVideoOutputRestoreState (  
    QTVideoOutputComponent    vo,  
    QTAtomContainer            state );
```

vo

The instance of a video output component for this request. Your software obtains this reference when calling *OpenComponent* (II–1130) or *OpenDefaultComponent* (II–1131).

state

A QT atom container, returned earlier by *QTVideoOutputSaveState* (II–1331), that contains state information for the video output component.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Discussion

If your software saves state information to disk, it must read the QT atom container structure from disk before calling this function. When your software restores state information for a video output component, the current display mode may change.

Because of this, your software must call this function before calling `QTVideoOutputBegin` (II–1321).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Saving and Restoring Component Configurations” (V–2894)

QTVideoOutputSaveState

Saves state information for an instance of a video output component.

```
ComponentResult QTVideoOutputSaveState (
    QTVideoOutputComponent    vo,
    QTAtomContainer            *state );
```

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

state

A pointer to complete information about the video output component’s current configuration.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

When your software saves state information for an instance of a video output component, it can restore this information when reconnecting to the component by calling `QTVideoOutputRestoreState` (II–1330).

Special Considerations

When your software calls this function, it must dispose of the QT atom container returned by the function by calling `QTDisposeAtomContainer` (II–1164).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Saving and Restoring Component Configurations” (V–2894)

QTVideoOutputSetClientName

QTVideoOutputSetClientName

Registers the name of an application or other software with an instance of a video output component.

```
ComponentResult QTVideoOutputSetClientName (
    QTVideoOutputComponent    vo,
    ConstStr255Param          str );
```

vo

The instance of a video output component for the request. Your software obtains this reference when it calls `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

str

The name of the application or other software to be registered.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The name you specify with this function can later be used by `QTVideoOutputGetCurrentClientName` (II–1325) to specify which software has exclusive access to the video output device controlled by the component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Registering the Name of Video Output Software” (V–2892)

See Also

For the corresponding get function, see `QTVideoOutputGetClientName` (II–1323).

QTVideoOutputSetDisplayMode

Specifies the display mode to be used by a video output component.

```
ComponentResult QTVideoOutputSetDisplayMode (
    QTVideoOutputComponent    vo,
    long                      displayModeID );
```

vo

The instance of a video output component for the request. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

displayModeID

The ID of the display mode to use. The ID specifies a QT atom of type `kQTVODisplayModeItem` in the QT atom container returned by `QTVideoOutputGetDisplayModeList` (II–1326).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

When software changes the display mode with this function, the change does not take effect until the next time the software calls `QTVideoOutputBegin` (II–1321) for the video output component. This lets the software change other output settings before displaying the video.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling the Video Output Display Mode” (V–2892)

See Also

For the corresponding get function, see `QTVideoOutputGetDisplayMode` (II–1325).

QTVideoOutputSetEchoPort

Specifies a window on the desktop in which to display video sent to the device.

```
ComponentResult QTVideoOutputSetEchoPort (
    QTVideoOutputComponent  vo,
    CGrafPtr                 echoPort );
```

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

echoPort

The window on the computer’s desktop in which to display the video.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

QTVRAnglesToCoord

Discussion

When your software sends video to the window you specify, the video is both displayed in the window and sent to the normal output of the video output device. When an output device can display video both on an external video display and in a window on a computer's desktop, the video displayed on the desktop is often at a smaller size and / or lower frame rate.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: "Controlling Video Output" (V-2893)

QTVRAnglesToCoord

Obtains a floating-point coordinate determined by a pair of pan and tilt angles.

```
OSErr QTVRAnglesToCoord (
    QTVRInstance      qtvr,
    float              panAngle,
    float              tiltAngle,
    QTVRFloatPoint     *coord );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II-1373).

panAngle

A pan angle.

tiltAngle

A tilt angle.

coord

On entry, a pointer to a `QTVRFloatPoint` (IV-2396) structure. On return, that structure is set to the coordinate of the specified movie that corresponds to the specified pan and tilt angles.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Special Considerations

`QTVRAnglesToCoord` is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Converting Angles and Points” (V–2911)

See Also

Use QTVRCoordToAngles (II–1338) to get a pair of pan and tilt angles from a floating-point coordinate.

QTVRBeginUpdateStream

Begins a stream of immediate updates to a QuickTime VR movie.

```
OSErr QTVRBeginUpdateStream (
    QTVRInstance      qtvr,
    QTVRImagingMode    imagingMode );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

imagingMode

An imaging mode (see below).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

imagingMode Constants

kQTVRStatic

The panorama is static

kQTVRMotion

The panorama is in motion; that is, an action such as panning or zooming is occurring.

kQTVRA11Modes

All currently defined imaging modes. You can specify this imaging mode when calling QTVRSetImagingProperty (II–1422) to assign a value to a particular imaging property for all imaging modes. You can also use this imaging mode in the *imagingMode* field of a QTVRPanoImagingAtom (IV–2398) structure to indicate a default value for a particular imaging property for all imaging modes.

Discussion

This function configures the QuickTime VR movie specified by the *qtvr* parameter for a stream of immediate updates to its movie image. After calling QTVRBeginUpdateStream, you perform the updates by calling QTVRUpdate (II–1445). When you are finished performing the updates, call QTVREndUpdateStream (II–1343).

QTVRCallInterceptedProc

Issuing a stream of image updates in this manner is significantly faster than simply calling QTVRUpdate repeatedly (that is, not within a begin/end update sequence). Each call to this function must be balanced by a call to QTVREndUpdateStream, but you can nest these calls.

Special Considerations

After you call this function and before you call QTVREndUpdateStream (II–1343), you must not change any of the QuickTime VR movie’s imaging properties.

Calling this function locks down large blocks of memory. As a result, you should minimize the amount of time before calling QTVREndUpdateStream.

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Imaging Characteristics” (V–2910)

See Also

Use QTVREndUpdateStream (II–1343) to end a stream of immediate updates to a QuickTime VR movie.

QTVRCallInterceptedProc

Calls an intercepted QuickTime VR function from within an intercept procedure.

```
OSErr QTVRCallInterceptedProc (
    QTVRInstance      qtvr,
    QTVRInterceptRecord *qtvrMsg );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

qtvrMsg

A pointer to a QTVRInterceptRecord (IV–2396) structure that specifies the function that your procedure is intercepting and the parameters for that function. This should be the same intercept record passed to your intercept procedure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function executes the QuickTime VR Manager function indicated by the `selector` field of the `qtvMsg` intercept record. The parameters passed to that function are the QuickTime VR movie specified by the `qtv` parameter and any other parameters contained in the `parameter` field of the `qtvMsg` record. You can, if you wish, change the parameters in that field before calling this function.

You can call this function more than once in your intercept procedure. In addition, the QuickTime VR Manager will call the intercepted function again unless your intercept procedure returns `TRUE` in the `cancel` parameter.

Special Considerations

You should call `QTVRCallInterceptedProc` only in an intercept procedure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Intercepting QuickTime VR Manager Routines” (V-2908)

Related Java Methods

`quicktime.vr.QTVRInstance.callInterceptedProc()`

QTVRColumnToPan

Get the pan angle that corresponds to a column number in the object image array.

```
float QTVRColumnToPan (
    QTVRInstance  qtv,
    short         column );
```

qtv

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II-1373).

column

A column number.

function result The pan angle that corresponds to the zero-based column number in the object image array specified by the `column` parameter.

Special Considerations

This function is valid only for object nodes.

QTVRCoordToAngles

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Converting Angles and Points” (V–2911)

Related Java Methods

`quicktime.vr.QTVRInstance.columnToPan()`

QTVRCoordToAngles

Get the pan and tilt angles of a floating-point coordinate in a panorama.

```
OSErr QTVRCoordToAngles (
    QTVRInstance      qtvr,
    QTVRFloatPoint    *coord,
    float              *panAngle,
    float              *tiltAngle );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

coord

On entry, a pointer to a `QTVRFloatPoint` (IV–2396) structure that specifies a coordinate in the full panorama.

panAngle

On entry, a pointer to a floating-point value. On return, that value contains the pan angle of the specified coordinate.

tiltAngle

On entry, a pointer to a floating-point value. On return, that value contains the tilt angle of the specified coordinate.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns, in the floating-point values pointed to by the `panAngle` and `tiltAngle` parameters, the pan and tilt angles of the point specified by the `coord` parameter. This function is useful for setting up angles in a back buffer imaging procedure; if you know a coordinate in the back buffer, you can call `QTVRCoordToAngles` to get the corresponding angles.

Special Considerations

QTVRCoordToAngles is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Converting Angles and Points” (V–2911)

See Also

Use QTVRAnglesToCoord (II–1334) to get a floating-point coordinate from a pair of pan and tilt angles.

QTVREnableFrameAnimation

Enables or disables frame animation for an object node.

```
OSErr QTVREnableFrameAnimation (
    QTVRInstance  qtvr,
    Boolean       enable );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

enable

A Boolean value that indicates whether to enable (TRUE) or disable (FALSE) frame animation for the specified object node.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function enables or disables the frame animation state for the object node specified by the *qtvr* parameter, according to the value of the *enable* parameter. The current frame rate, set by the function QTVRSetFrameRate (II–1421), is unaffected by the state of frame animation of an object node.

Special Considerations

This function is valid only for object nodes. You should use this function instead of standard QuickTime functions to control object animation.

Version Notes

Introduced in QuickTime 3 or earlier.

QTVREnableHotSpot

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use `QTVRGetFrameAnimation` (II–1361) to get the current state of frame animation for an object node.

QTVREnableHotSpot

Enables or disables one or more QTVR hot spots.

```
OSErr QTVREnableHotSpot (
    QTVRInstance    qtvr,
    UInt32          enableFlag,
    UInt32          hotSpotValue,
    Boolean          enable );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

enableFlag

The kind of hot spot or hot spots to enable or disable (see below).

hotSpotValue

The desired hot spot or spots, defined by the specified enabled flag (see below).

enable

A Boolean value that indicates whether the specified hot spots are to be enabled (TRUE) or disabled (FALSE).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

enableFlag Constants

`kQTVRHotSpotID`

The `hotSpotValue` parameter specifies a hot spot ID.

`kQTVRHotSpotType`

The `hotSpotValue` parameter specifies a hot spot type.

`kQTVRA11HotSpots`

Set the `hotSpotValue` parameter to 0.

Discussion

This function either enables or disables the hot spot or spots specified by the `enableFlag` and `hotSpotValue` parameters, according to the value of the `enable`

parameter. The hot spots are always selected from among the hot spots in the current node of the QuickTime VR movie specified by the `qtvr` parameter.

Normally, all hot spots in a node are enabled (that is, the cursor automatically changes shape when it is moved over a hot spot, and the `QTVRTriggerHotSpot` (II-1443) function is called internally when the user clicks a hot spot). When a hot spot is disabled, QuickTime VR behaves as if the hot spot were not present.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Hot Spots” (V-2906)

Related Java Methods

`quicktime.vr.QTVRInstance.enableHotSpot()`

QTVREnableTransition

Enables or disables a transition effect.

```
OSErr QTVREnableTransition (
    QTVRInstance  qtvr,
    UInt32        transitionType,
    Boolean        enable );
```

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II-1373).

`transitionType`

A type of transition property (see below). Currently only one constant is available for this parameter.

`enable`

A `Boolean` value that indicates whether the specified transition property is to be enabled (`TRUE`) or disabled (`FALSE`).

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

transitionType Constant

`kQTVRTransitionSwing`

A transition between two views in a single node.

QTVREnableViewAnimation

Discussion

This function enables or disables the transition property specified by the `transitionType` parameter for the movie specified by the `qtvr` parameter, as indicated by the value of the `enable` parameter. Once a transition effect is enabled, it is used at the appropriate time until it is disabled by a subsequent call to this function.

Special Considerations

`QTVREnableTransition` is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Imaging Characteristics” (V–2910)

See Also

Use `QTVRSetTransitionProperty` (II–1435) to set the value of a transition property.

QTVREnableViewAnimation

Enables or disables view animation for an object node.

```
OSErr QTVREnableViewAnimation (
    QTVRInstance  qtvr,
    Boolean       enable );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

enable

A Boolean value that indicates whether to enable (TRUE) or disable (FALSE) view animation for the specified object node.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You should use this function instead of standard QuickTime functions to control object animation.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use QTVRGetViewAnimation (II–1377) to get the current state of view animation for an object node.

QTVREndUpdateStream

Ends a stream of immediate updates to a QuickTime VR movie.

```
OSErr QTVREndUpdateStream (
    QTVRInstance  qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function unlocks the memory locked by the matching call to QTVRBeginUpdateStream (II–1335) for the QuickTime VR movie specified by the *qtvr* parameter and reverses any other actions performed by that call. Each call to QTVRBeginUpdateStream must be balanced by a call to this function, but you can nest these calls. For nested calls, only the final call to this function unlocks the memory locked by the first call to QTVRBeginUpdateStream.

Special Considerations

QTVREndUpdateStream is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Imaging Characteristics” (V–2910)

See Also

Use QTVRBeginUpdateStream (II–1335) to begin a stream of immediate updates to a QuickTime VR movie.

QTVRGetAngularUnits

QTVRGetAngularUnits

Obtains the type of unit currently used when specifying angles.

```
QTVRAngularUnits QTVRGetAngularUnits (  
    QTVRInstance    qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

function result The type of unit currently used (see below).

Function Result Constants

kQTVRDegrees

Angles are specified using degrees. This is the default type of angle specification.

kQTVRRadians

Angles are specified using radians

Discussion

This function returns, as its function result, a constant that indicates the type of angular unit currently used by the movie instance specified by the qtvr parameter. Angular values you pass to other QuickTime VR functions, such as QTVRSetPanAngle (II–1431), are interpreted in those units.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Converting Angles and Points” (V–2911)

Related Java Methods

```
quicktime.vr.QTVRInstance.getAngularUnits()
```

See Also

Use QTVRSetAngularUnits (II–1408) to set the type of angular unit used by a QuickTime VR movie.

QTVRGetAnimationSetting

Obtains the current state of an animation setting for an object node.

```
OSErr QTVRGetAnimationSetting (
```

```
QTVRInstance      qtvr,
QTVRObjectAnimationSetting setting,
Boolean           *enable );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II-1373).

setting

An animation setting (see below).

enable

On entry, a pointer to a Boolean value. On return, that value is set to TRUE if the specified animation setting is currently enabled for the specified object node or to FALSE otherwise.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

setting Constants

kQTVRPalindromeViewFrames

Play a back-and-forth animation of the frames of the current view. The frames of the current view play with a positive or negative frame rate; the frame rate sign is switched each time the view end time (equal to the view duration) or the view start time (always 0) is reached.

kQTVRStartFirstViewFrames

Play the frame animation starting with the first frame in the view (that is, at the view start time). This setting is useful if a sound track is associated by time with object views. Even if the object view contains no animation, setting this flag allows any sound authored to play with the view to play from the beginning. When this flag is clear, each new object view begins playing using the current view time of the previous view.

kQTVRDontLoopViewFrames

Don’t loop the frame animation. Animation frames and sound stop playing when the view duration is reached.

kQTVRPlayEveryViewFrame

Play every view frame. Animation plays all frames regardless of play rate. The play rate is used to adjust the duration in which a frame appears but no frames are skipped so the rate is not exact. When this property is set, sound tracks are not played.

kQTVRSyncViewToFrameRate

Synchronize the view animation to the frame animation. When view animation is enabled, the object views play at the same rate and animation settings as the frame animation rate and settings. This is useful if animation must be

QTVRGetAvailableResolutions

synchronized precisely across multiple views or a sound track is to be played during view animation instead of during frame animation.

kQTVRPalindromeViews

Play a back-and-forth animation of the views of the current node. When view animation is enabled, the object views play with a positive or negative view rate; the view rate sign is switched each time an object's pan equals the object's minimum pan limit or the object's maximum pan limit (the first column in a row or the last column in a row, respectively).

kQTVRPlayStreamingViews

When an object movie is streaming in from a network, this flag indicates whether the frames corresponding to the row in which the default view appears are to be played as they are loaded.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use `QTVRSetAnimationSetting` (II–1409) to set the state of an animation setting for an object node.

QTVRGetAvailableResolutions

Obtains the image resolutions present in the current node.

```
OSErr QTVRGetAvailableResolutions (
    QTVRInstance    qtvr,
    UInt16          *resolutionsMask );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

resolutionsMask

On entry, a pointer to an unsigned short integer. On return, that integer is set to a bitmask that encodes the image resolutions (see below) available at the current node.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

resolutionsMask Constants

kQTVRDefaultRes

The default resolution of the image.

kQTVRFullRes

The full resolution of the image.

kQTVRHalfRes

One-half the full resolution of the image.

kQTVRQuarterRes

One-quarter the full resolution of the image.

Discussion

A single node can contain multiple resolutions of a panorama or an object. The lowest order bit is always set and corresponds to the base resolution of the node. Each succeeding bit corresponds to a resolution that is half that (both horizontally and vertically) of the preceding bit. If an image with a corresponding resolution is present in the current node, that bit is set.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing VR Memory” (V–2913)

QTVRGetBackBufferMemInfo

Obtains information about the internal back buffer that QuickTime VR maintains for caching panoramic images.

```
OSErr QTVRGetBackBufferMemInfo (
    QTVRInstance    qtvr,
    UInt32          geometry,
    UInt16          resolution,
    UInt32          cachePixelFormat,
    SInt32          *minCacheBytes,
    SInt32          *suggestedCacheBytes,
    SInt32          *fullCacheBytes );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

QTVRGetBackBufferMemInfo

geometry

The geometry parameter (see below) specifies the type and orientation of the panorama data.

resolution

The resolution for which the information is desired (see below).

cachePixelFormat

The desired pixel format for the back buffer. This value should be one of the defined pixel formats (see below).

minCacheBytes

On entry, a pointer to a long integer. On return, that long integer is set to the minimum size, in bytes, of the back buffer required to display the specified panorama with a severely limited maximum field of view. Set this parameter to NIL to prevent this information from being returned.

suggestedCacheBytes

On entry, a pointer to a long integer. On return, that long integer is set to the minimum size, in bytes, of the back buffer required to display the specified panorama with full wide-angle zooming. Set this parameter to NIL to prevent this information from being returned.

fullCacheBytes

On entry, a pointer to a long integer. On return, that long integer is set to the minimum size, in bytes, of the back buffer required to have the entire panorama in memory at once. That is the default size of the panorama back buffer. Set this parameter to NIL to prevent this information from being returned.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

geometry Constants

`kQTVRUseMovieGeometry`

Value is 0.

`kQTVRVerticalCylinder`

Vertical cylinder geometry. Value is 'vcyl'.

resolution Constants

`kQTVRDefaultRes`

The default resolution of the image.

`kQTVRFullRes`

The full resolution of the image.

`kQTVRHalfRes`

One-half the full resolution of the image.

`kQTVRQuarterRes`

One-quarter the full resolution of the image.

cachePixelFormat Constants

`kQTVRMinimumCache`

The minimum cache size required to display the specified movie.

`kQTVRSuggestedCache`

The suggested cache size, a cache large enough to allow full zooming out of the panorama.

`kQTVRFullCache`

The full cache size (that is, a cache that is large enough to fit the entire panorama in memory). This is the default cache size.

Discussion

You can use this function to get information about the size of the back buffer that would be required for caching a panoramic image of a specified pixel format, geometry, and resolution. This is a “what-if” function: you specify a resolution and a pixel format, and this function returns several buffer sizes. You can use this information, in conjunction with `QTVRSetBackBufferPrefs` (II–1412), to exercise some control over the size of the back buffer.

The resolution at which an image is to be displayed is specified by the `resolution` parameter. You can use a resolution that is not in the movie file. Relative to that resolution and the pixel depth determined by the `cachePixelFormat` parameter, this function returns, through the `minCacheBytes` parameter, the minimum size of the buffer needed to display the movie. Using a buffer of that size, however, may result in a severely limited maximum field of view. You can call the `QTVRGetViewingLimits` (II–1379) function to determine the actual maximum field of view.

To allow full wide-angle zooming, you should use a buffer whose size is specified by either the `suggestedCacheBytes` parameter or the `fullCacheBytes` parameter.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing VR Memory” (V–2913)

QTVRGetBackBufferSettings

QTVRGetBackBufferSettings

Obtains information about the resolution, pixel format, and size of the back buffer maintained internally by QuickTime VR for caching a panoramic image in a particular pixel format.

```
OSErr QTVRGetBackBufferSettings (
    QTVRInstance    qtvr,
    UInt32          *geometry,
    UInt16          *resolution,
    UInt32          *cachePixelFormat,
    SInt16          *cacheSize );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

geometry

The type and orientation of the panorama data (see below).

resolution

On entry, a pointer to an unsigned short integer. On return, that integer is set to the index of the current image resolution (see below).

cachePixelFormat

On entry, a pointer to a long integer. On return, that long integer is set to the pixel format of the current panorama back buffer (see below).

cacheSize

On entry, a pointer to a short integer. On return, that integer is set to a value that describes the size of the current panorama back buffer.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

geometry Constants

`kQTVRUseMovieGeometry`

Value is 0.

`kQTVRVerticalCylinder`

Vertical cylinder geometry. Value is 'vcyl'.

resolution Constants

`kQTVRDefaultRes`

The default resolution of the image.

`kQTVRFullRes`

The full resolution of the image.

kQTVRHalfRes

One-half the full resolution of the image.

kQTVRQuarterRes

One-quarter the full resolution of the image.

cachePixelFormat Constants

k16LE555PixelFormat

16-bit LE RGB 555 (Windows)

k16BE565PixelFormat

16-bit BE RGB 565

k16LE565PixelFormat

16-bit LE RGB 565

k24BGRPixelFormat

24-bit BGR

k32ARGBPixelFormat

32-bit ARGB

k32BGRAPixelFormat

32-bit BGRA (Matrox)

k32ABGRPixelFormat

32-bit ABGR

k32RGBAPixelFormat

32-bit RGBA

cacheSize Constants

kQTVRMinimumCache

The minimum cache size required to display the specified VR movie.

kQTVRSuggestedCache

The suggested cache size, a cache large enough to allow full zooming out of the panorama.

kQTVRFullCache

The full cache size (that is, a cache that is large enough to fit the entire panorama in memory). This is the default cache size.

Discussion

This function returns, through the `resolution` parameter, the index of the current resolution for the QuickTime VR movie specified by the `qtvr` parameter. The index indicates which bit in the mask value returned by `QTVRGetAvailableResolutions` (II-1346) specifies the current resolution. For example, if the returned index is 1, the base resolution is being used. If the returned index is 2, then a resolution of half the



QTVRGetConstraints

base resolution is being used. This function also returns the pixel format and the cache size in the `cachePixelFormat` and `cacheSize` parameters, respectively.

The QuickTime VR file might not contain an image track corresponding to the resolution indicated by the resolution value returned. The QuickTime VR Manager may have set a lower resolution because memory is low, or the resolution may have been set by a call to `QTVRSetBackBufferPrefs` (II–1412).

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing VR Memory” (V–2913)

See Also

Use `QTVRSetBackBufferPrefs` (II–1412) to set the resolution, cache depth, and cache size of the panorama back buffer maintained internally by QuickTime VR for caching an image. Use `QTVRGetAvailableResolutions` (II–1346) to determine which resolutions are supported by a node. Use `QTVRGetBackBufferMemInfo` (II–1347) to determine the memory requirements for the preferred settings.

QTVRGetConstraints

Obtains the current constraints of a QuickTime VR movie.

```
OSErr QTVRGetConstraints (
    QTVRInstance    qtvr,
    UInt16          kind,
    float            *minValue,
    float            *maxValue );
```

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

`kind`

The type of constraints to be returned (see below).

`minValue`

On entry, a pointer to a floating-point value. On return, the current minimum constraint of the specified type is copied into that value.

`maxValue`

On entry, a pointer to a floating-point value. On return, the current maximum constraint of the specified type is copied into that value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

kind Constants

`kQTVRPan`

The pan angle. The associated value is of type `float`.

`kQVRTilt`

The tilt angle. The associated value is of type `float`.

`kQTVRFieldOfView`

The field of view. The associated value is of type `float`.

Discussion

This function returns, in the floating-point values pointed to by the `minValue` and `maxValue` parameters, the current minimum and maximum constraints of the type specified by the `kind` parameter. The values returned by this function are unaffected by the current control settings.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Determining Viewing Limits and Constraints” (V–2912)

Related Java Methods

`quicktime.vr.QTVRInstance.getConstraints_min()`,

`quicktime.vr.QTVRInstance.getConstraints_max()`

See Also

For the corresponding set function, see `QTVRSetConstraints` (II–1415), used to set a movie’s minimum and maximum constraints.

QTVRGetConstraintStatus

Obtains the set of constraints active for the current view.

```
UInt32 QTVRGetConstraintStatus (
    QTVRInstance  qtvr );
```

QTVRGetConstraintStatus

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II-1373).

function result A long integer (see below) whose bits encode the constraints currently active for the QuickTime VR movie specified by the qtvr parameter.

Function Result Constants

kQTVRUnconstrained

The view has no constraints. This is not a bit selector but a value that indicates that none of the other bits are set.

kQTVRCantPanLeft

If this bit is set, the view cannot pan left.

kQTVRCantPanRight

If this bit is set, the view cannot pan right.

kQTVRCantPanUp

If this bit is set, the view cannot pan up.

kQTVRCantPanDown

If this bit is set, the view cannot pan down.

kQTVRCantZoomIn

If this bit is set, the view cannot zoom in.

kQTVRCantZoomOut

If this bit is set, the view cannot zoom out.

kQTVRCantTranslateLeft

If this bit is set, the view cannot translate to the left. This constraint is valid only for object nodes.

kQTVRCantTranslateRight

If this bit is set, the view cannot translate to the right. This constraint is valid only for object nodes.

kQTVRCantTranslateUp

If this bit is set, the view cannot translate up. This constraint is valid only for object nodes.

kQTVRCantTranslateDown

If this bit is set, the view cannot translate down. This constraint is valid only for object nodes.

Discussion

The values returned by `QTVRGetConstraintStatus` are unaffected by the current control settings.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Determining Viewing Limits and Constraints” (V–2912)

QTVRGetControlSetting

Obtains the current state of a control setting for an object node.

```
OSErr QTVRGetControlSetting (
    QTVRInstance      qtvr,
    QTVRControlSetting setting,
    Boolean            *enable );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

setting

A control setting (see below).

enable

On entry, a pointer to a Boolean value. On return, that value is set to `TRUE` if the specified control setting is currently enabled for the specified object node or to `FALSE` otherwise.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

setting Constants

`kQTVRWrapPan`

Set wrapping during panning. When this control setting is enabled, the user can wrap around from the current pan constraint maximum value to the pan constraint minimum value (or vice versa) using the mouse or arrow keys. In addition, calling `QTVRSetPanAngle` (II–1431) with a pan angle that is either less than or greater than the current pan constraints results in a pan angle within the current pan constraints. Similarly, `QTVRWrapAndConstrain` (II–1446) returns a pan angle within the current pan constraints, and `QTVRNudge` (II–1402) wraps views after it reaches the maximum or minimum constraint value.

QTVRGetControlSetting

When this control setting is disabled, the user cannot wrap around from the current pan constraint maximum value to the pan constraint minimum value (or vice versa) using the mouse or arrow keys. In addition, `QTVRSetPanAngle` returns the result code `constraintReachedErr` when passed a pan angle that is either less than or greater than the current pan constraints. Similarly, `QTVRWrapAndConstrain` returns a pan angle that is one of the current pan constraints, and `QTVRNudge` returns the result code `constraintReachedErr` when it reaches the maximum or minimum constraint value.

Note that a view animation stops when a constraint is reached unless palindrome view animation or wrapping during panning is enabled.

kQTVRWrapTilt

Set wrapping during tilting. When this control setting is enabled, the user can wrap around from the current tilt constraint maximum value to the tilt constraint minimum value (or vice versa) using the mouse or arrow keys. In addition, calling `QTVRSetTiltAngle` (II–1434) with a tilt angle that is either less than or greater than the current tilt constraints results in a tilt angle within the current tilt constraints. Similarly, `QTVRWrapAndConstrain` (II–1446) returns a tilt angle within the current tilt constraints, and `QTVRNudge` (II–1402) wraps views after it reaches the maximum or minimum constraint value.

When this control setting is disabled, the user cannot wrap around from the current tilt constraint maximum value to the tilt constraint minimum value (or vice versa) using the mouse or arrow keys. In addition, the `QTVRSetTiltAngle` function returns the result code `constraintReachedErr` when passed a tilt angle that is either less than or greater than the current tilt constraints. Similarly, `QTVRWrapAndConstrain` returns a tilt angle that is one of the current tilt constraints, and `QTVRNudge` returns the result code `constraintReachedErr` when it reaches the maximum or minimum constraint value.

kQTVRCanZoom

Set zooming to be active. When this control setting is enabled, the user can change the current field of view using the zoom-in and zoom-out keys on the keyboard (or using the VR controller buttons). In addition, you can use `QTVRSetFieldOfView` (II–1420) to alter the current field of view. Similarly, `QTVRWrapAndConstrain` (II–1446) returns a field of view within the current field of view constraints.

When this control is disabled, the user cannot change the current field of view using the zoom-in and zoom-out keys on the keyboard (or using the VR controller buttons). In addition, the `QTVRSetFieldOfView` function returns the

result code `constraintReachedErr` and doesn't change the field of view. Similarly, `QTVRWrapAndConstrain` returns the current field of view.

kQTVRReverseHControl

Reverse the direction of the horizontal control. When this control setting is enabled, the user can change an object's horizontal view using the mouse or the keyboard arrows with reversed values for mouse horizontal motion and keyboard left and right arrows. (In other words, the left arrow key causes panning to the right, and moving the mouse right causes panning to the left.) Similarly, `QTVRNudge` (II-1402) nudges left when you pass the value 0 and right when you pass the value 180. This control setting is useful when an object's frames have been authored in reverse order.

kQTVRReverseVControl

Reverse the direction of the vertical control. When this control setting is enabled, the user can change an object's vertical view using the mouse or the keyboard arrows with reversed values for mouse vertical motion and keyboard up and down arrows. (In other words, the up arrow key causes tilting down, and moving the mouse down causes tilting up.) Similarly, `QTVRNudge` (II-1402) nudges up when you pass the value 270 and down when you pass the value 90. This control setting is useful when an object's frames have been authored in reverse order.

kQTVRSwapHVControl

Exchange the horizontal and vertical controls. When this setting is enabled, the user can pan left using the up arrow key and tilt up using the left arrow key. Similarly, `QTVRNudge` (II-1402) nudges up when you pass the value 180 and down when you pass the value 0. This control setting is useful if an object is a single-row or multirow movie containing vertically changing images. This is especially useful when enabling view animation for an object with animating vertical changes in view (because objects will animate only along rows, not along columns).

kQTVRTranslation

Set translation to be active. When this setting is enabled, the user can translate using the mouse when either the translation key is held down or the controller translation mode button is toggled on. In addition, you can use the `QTVRSetViewCenter` function to set a horizontal and vertical position in the current display coordinate system. When this setting is disabled, `QTVRWrapAndConstrain` (II-1446) always returns the current view center, and `QTVRSetViewCenter` (II-1437) returns the result code `constraintReachedErr` and doesn't change the current view center.

QTVRGetCurrentMouseMode

Discussion

This function returns, through the `enable` parameter, the current state of the control setting specified by the `setting` parameter for the object node specified by the `qtv` parameter. If `enable` is `TRUE`, the specified setting is currently enabled; otherwise, the setting is disabled.

Special Considerations

`QTVRGetControlSetting` is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use `QTVRSetControlSetting` (II–1416) to set the state of a control setting for an object node.

QTVRGetCurrentMouseMode

Obtains the current mouse control modes.

```
UInt32 QTVRGetCurrentMouseMode (
    QTVRInstance    qtv );
```

qtv

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

function result A constant (see below) that describes the current mouse control modes.

Function Result Constants

`kQTVRPanning`

If this bit is set, the mouse controls panning of standard objects, using objects-only controllers.

`kQTVRTranslating`

If this bit is set, the mouse controls translation for all objects.

`kQTVRZooming`

If this bit is set, the mouse controls zooming for all objects.

kQTVRScrolling

If this bit is set, the mouse controls arrow scrolling for standard objects and scrolling for joystick objects.

kQTVRSelecting

If this bit is set, the mouse controls selecting of objects as an object absolute controller.

Discussion

The value returned by this function is an unsigned long integer that encodes the current mouse control modes. If a bit in the integer is set, the corresponding mode is one of the current mouse modes. The mode bits are addressed using the above constants. Notice that several modes can be returned. That means a return value could have both zooming and translating set, for example.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

QTVRGetCurrentNodeID

Obtains the current node of a movie.

```
UInt32 QTVRGetCurrentNodeID (
    QTVRInstance  qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

function result The ID of the current node of the specified movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Getting Scene and Node Information” (V–2906)

Related Java Methods

quicktime.vr.QTVRInstance.getCurrentNodeID()

QTVRGetCurrentViewDuration

See Also

Use QTVRGoToNodeID (II–1386) to set the current node ID.

QTVRGetCurrentViewDuration

Obtains the duration of the current view of an object node.

```
TimeValue QTVRGetCurrentViewDuration (  
    QTVRInstance    qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

function result The duration of the current view of the object node specified by the *qtvr* parameter.

Special Considerations

This function is valid only for object nodes. You cannot change a node's view duration.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

QTVRGetFieldOfView

Obtains the vertical field of view of a QuickTime VR movie.

```
float QTVRGetFieldOfView (  
    QTVRInstance    qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

function result The current vertical field of view of the QuickTime VR movie specified by the *qtvr* parameter. The vertical field of view is a floating-point value that specifies the angle created by the two lines that connect the viewpoint to the top and bottom of the image.

Discussion

The following code fragment illustrates the use of this function:

```
// QTVRGetFieldOfView coding example
#define kDirIn      4L
#define kDirOut     5L

void MyZoomInOrOut (QTVRInstance theInstance, long theDir)
{
    float    theFloat;
    theFloat = QTVRGetFieldOfView(theInstance);
    switch (theDir) {
        case kDirIn:
            theFloat = theFloat / 2.0;
            break;
        case kDirOut:
            theFloat = theFloat * 2.0;
            break;
        default:
            break;
    }
    QTVRSetFieldOfView(theInstance, theFloat);
    QTVRUpdate(theInstance, kQTVRStatic);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Manipulating Viewing Angles and Zooming” (V-2905)

Related Java Methods

quicktime.vr.QTVRInstance.getFieldOfView()

See Also

Use QTVRSetFieldOfView (II-1420) to set the vertical field of view of a QuickTime VR movie.

QTVRGetFrameAnimation

Obtains the current state of frame animation for an object node.

QTVRGetFrameRate

```
Boolean QTVRGetFrameAnimation (  
    QTVRInstance    qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

function result TRUE if frame animation is currently enabled, FALSE otherwise.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use QTVREnableFrameAnimation (II–1339) to set the state of frame animation for an object node.

QTVRGetFrameRate

Obtains the current frame rate of an object node.

```
float QTVRGetFrameRate (  
    QTVRInstance    qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

function result The current frame rate of the object node specified by the qtvr parameter. A frame rate is a floating-point value in the range from –100.0 to +100.0.

Discussion

An object node’s default frame rate is stored in the movie file.

Special Considerations

QTVRGetFrameRate is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use QTVRSetFrameRate (II–1421) to set the frame rate of an object node.

QTVRGetHotSpotRegion

Obtains the region occupied by a hot spot.

```
OSErr QTVRGetHotSpotRegion (
    QTVRInstance    qtvr,
    UInt32          hotSpotID,
    RgnHandle        hotSpotRegion );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

hotSpotID

A hot spot ID.

hotSpotRegion

On entry, an initialized handle to a region. On return, this region is rewritten with the region occupied by the hot spot having the specified ID.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The returned region is clipped to the bounds of the movie’s graphics world. You can obtain the regions of all visible hot spots by calling QTVRGetVisibleHotSpots (II–1384) and then calling this function for each hot spot ID in the list.

Special Considerations

The first time you call this function, a significant amount of memory might need to be allocated. Accordingly, you should always check for Memory Manager errors returned by this function. Your application is responsible for disposing of the memory occupied by the returned region.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Hot Spots” (V–2906)

QTVRGetHotSpotType

QTVRGetHotSpotType

Obtains the type of a QuickTime VR hot spot.

```
OSErr QTVRGetHotSpotType (
    QTVRInstance    qtvr,
    UInt32          hotSpotID,
    OSType          *hotSpotType );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

hotSpotID

A hot spot ID.

hotSpotType

On entry, a pointer to a long integer. On return, that long integer contains the type of the hot spot specified by the hot spot ID.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function gets the type of a hot spot whose ID you specify. In combination with the `kQTVRGetHotSpotTypeSelector` intercept selector (see `QTVRInstallInterceptProc` (II–1387)), this allows an application to change a hot spot’s type dynamically. For example, an application can take an existing movie and cause VR to display the cursors for a type of hotspot different from the one the movie was originally authored with. In combination with intercepting `kQTVRTriggerHotSpotSelector`, this would allow an Internet plugin to override undefined or link hotspots in movies to make them appear and act as though they are URL links instead. If `kQTVRTriggerHotSpotSelector` is not intercepted, VR will attempt to act on the hotspot in the normal way; by storing both link and URL data in a file, the exact behavior can be determined at runtime by an application to allow linking to either another node locally or a remote URL link, depending on system configuration or other arbitrary considerations.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Hot Spots” (V–2906)

Related Java Methods

`quicktime.vr.QTVRInstance.getHotSpotType()`

See Also

Use `kQTVRGetHotSpotTypeSelector` with `QTVRInstallInterceptProc` (II–1387) to set the selector for the intercept procedure.

QTVRGetImagingProperty

Obtains the current value of an imaging property of a movie.

```
OSErr QTVRGetImagingProperty (
    QTVRInstance      qtvr,
    QTVRImagingMode    imagingMode,
    UInt32             imagingProperty,
    SInt32             *propertyValue );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

imagingMode

An imaging mode (see below).

imagingProperty

An imaging property (see below).

propertyValue

On entry, a pointer to a long integer. On return, that long integer contains the current value of the specified imaging property for the specified mode.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

imagingMode Constants

`kQTVRStatic`

The panorama is static

`kQTVRMotion`

The panorama is in motion; that is, an action such as panning or zooming is occurring.

`kQTVRAllModes`

All currently defined imaging modes. You can specify this imaging mode when calling `QTVRSetImagingProperty` (II–1422) to assign a value to a particular imaging property for all imaging modes. You can also use this imaging mode in the `imagingMode` field of a `QTVRPanoImagingAtom` (IV–2398) structure to indicate a default value for a particular imaging property for all imaging modes.

QTVRGetImagingProperty

imagingProperty Constants

kQTVRImagingCorrection

The correction mode property (see below). This property determines the type of image correction to be applied when imaging a panoramic view.

kQTVRImagingQuality

The imaging quality property (see below). This property determines the quality of the output image.

kQTVRImagingDirectDraw

The direct-drawing property. This property determines the immediate destination of an image. The value of this property (as specified by the `propertyValue` parameter) is interpreted as a `Boolean` value. If the value is `TRUE`, then images are computed and drawn directly to the final destination without first being drawn in the prescreen buffer maintained by QuickTime VR. This value provides the fastest overall frame rate but may result in relatively slower individual frame drawing for high-quality, high-resolution images on lower performance systems. If the value of this property is `FALSE`, then images are computed and drawn first into the prescreen buffer and then to the final destination. This value provides the shortest imaging time for each individual frame but results in a reduced overall frame rate.

Note that the `kQTVRImagingDirectDraw` property cannot always be supported. During updates, QuickTime VR's warping engine might ignore a value of `TRUE` and draw the image first into the prescreen buffer and then into the final destination

kQTVRImagingCurrentMode

The current imaging mode property. When you pass this property selector to this function, it returns, in the `propertyValue` parameter, the current imaging mode (that is, either `kQTVRStatic` or `kQTVRMotion`). If the `kQTVRImagingDefaultValue` bit is set, the default imaging mode is returned. You can get but not set the value of this property. You might want to determine the current imaging mode in an imaging callback procedure if what you draw depends on the current imaging mode.

kQTVRImagingDefaultValue

The default value specifier. You can OR this value with any of the imaging property types to get or set the default value of that property type

kQTVRImagingCorrection Constants

kQTVRNoCorrection

Apply no warping. The source panorama is reproduced directly, with no image correction.

`kQTVRPartialCorrection`

Apply one-dimensional (horizontal) warping. This kind of correction is often sufficient, especially for outdoor scenes.

`kQTVRFullCorrection`

Apply two-dimensional warping. This correction mode produces images in correct perspective from the source panorama.

kQTVRImagingQuality Constants

`codecMinQuality`

The minimum valid value for a CodecQ field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a CodecQ field.

Discussion

This function returns, in the long integer pointed to by the `propertyValue` parameter, the current value of the property specified by the `imagingProperty` parameter when the QuickTime VR movie specified by the `qtvr` parameter is in the mode specified by the `imagingMode` parameter.

Special Considerations

`QTVRGetImagingProperty` is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Imaging Characteristics” (V–2910)

Related Java Methods

`quicktime.vr.QTVRInstance.getImagingProperty()`

See Also

Use `QTVRSetImagingProperty` (II–1422) to set an imaging property.

QTVRGetInteractionProperty

QTVRGetInteractionProperty

Obtains the value of an interaction property.

```
OSErr QTVRGetInteractionProperty (
    QTVRInstance  qtvr,
    UInt32        property,
    void          *value );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

property

An interaction property type (see below).

value

On entry, a pointer to a block of memory. On return, that memory contains the current value of the specified interaction property.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

property Constants

kQTVRInteractionMouseClickedHysteresis

The value parameter is interpreted as an unsigned short integer that represents the mouse-click hysteresis, the distance, in pixels, from the location of a mouse-down event to the limit within which the cursor is considered not to have moved. In other words, the cursor can move that many pixels during a mouse click and still have the event considered a click. The default mouse-click hysteresis value is 2. This property is valid for panoramas only.

kQTVRInteractionMouseClickedTimeout

The value parameter is interpreted as an unsigned long integer that represents the mouse-click timeout, the number of **ticks** after which a mouse click times out and is automatically switched from a hot spot selection into a pan. The default mouse-click timeout value is 30 ticks (one-half second). This property is valid for panoramas only.

kQTVRInteractionPanTiltSpeed

The panning and tilting speed. The value parameter is interpreted as an unsigned long integer that represents the relative speed of panning and tilting. This integer should be from 1 (the slowest speed) through 10 (the fastest speed); the default panning and tilting speed is 5. This property is valid only for panoramas.

kQTVRInteractionZoomSpeed

The value parameter is interpreted as an unsigned long integer that represents the zooming speed, the relative speed of zooming in and out. This integer should be from 1 (the slowest speed) through 10 (the fastest speed); the default zooming speed is 5. This property is valid for both objects and panoramas.

kQTVRInteractionTranslateOnMouseDown

The translate flag. The value parameter is interpreted as a Boolean value that indicates whether translate mode is enabled (true) or disabled (false). When translate mode is enabled, the user can no longer pan or tilt using the mouse; instead, dragging the cursor causes the object to translate. The default translate flag value is false. This property is valid for objects only.

kQTVRInteractionMouseMotionScale

The value parameter is interpreted as a pointer to a floating-point number that represents the mouse-motion scale, the number of degrees or radians that an object or panorama is panned or tilted when the cursor is dragged the entire width of the object bounding box. The default value is 180.0. This property is valid for objects only.

kQTVRInteractionNudgeMode

This parameter lets you set the QuickTime VR nudge mode (see below) to either rotate, translate, or be the same as the current mouse mode.

Nudge Mode Constants

kQTVRNudgeRotate

Set the nudge mode to rotate the object.

kQTVRNudgeTranslate

Set the nudge mode to translate the image.

kQTVRNudgeSameAsMouse

Set the nudge mode to the same as the current mouse mode, which you can determine by calling `QTVRGetCurrentMouseMode` (II–1358).

Discussion

This function returns, in the block of memory pointed to by the `value` parameter, the current value of the property specified by the `property` parameter for the QuickTime VR movie specified by the `qtvr` parameter. That block of memory must be large enough to hold the returned value.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing QuickTime VR Movie Interactions” (V–2912)

QTVRGetMouseDownTracking

See Also

Use `QTVRSetInteractionProperty` (II–1425) to set an interaction property.

QTVRGetMouseDownTracking

Obtains the current state of mouse-down tracking.

```
Boolean QTVRGetMouseDownTracking (  
    QTVRInstance    qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

function result A Boolean value that indicates whether QuickTime VR is currently handling mouse-down tracking for the QuickTime VR movie specified by the *qtvr* parameter (TRUE) or not (FALSE).

Discussion

By default, QuickTime VR tracks mouse clicks in a QuickTime VR movie and triggers hot spots as necessary.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Handling Events” (V–2907)

See Also

Use `QTVRSetMouseDownTracking` (II–1429) to change the mouse-down tracking state of a QuickTime VR movie.

QTVRGetMouseOverTracking

Obtains the current state of mouse-over tracking.

```
Boolean QTVRGetMouseOverTracking (  
    QTVRInstance    qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

function result A Boolean value that indicates whether QuickTime VR is currently handling mouse-over tracking for the QuickTime VR movie specified by the `qtvr` parameter (TRUE) or not (FALSE).

Discussion

By default, QuickTime VR tracks mouse movements in a QuickTime VR movie and changes the shape of the cursor as appropriate.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Handling Events” (V–2907)

See Also

Use `QTVRSetMouseOverTracking` (II–1431) to change the mouse-over tracking state of a QuickTime VR movie.

QTVRGetNodeInfo

Obtains the node information atom container that describes a node and all the hot spots in the node.

```
OSErr QTVRGetNodeInfo (
    QTVRInstance      qtvr,
    UInt32            nodeID,
    QTAtomContainer    *nodeInfo );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

nodeID

A node ID. Set this parameter to `kQTVRCurrentNode` to receive information about the current node.

nodeInfo

On return, a pointer to an atom container that contains information about the specified node.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns, in the `nodeInfo` parameter, a pointer to an atom container that contains information about the node specified by the `nodeID` parameter in the movie

QTVRGetNodeType

specified by the `qtv` parameter. The atom container includes information about all the hot spots contained in that node. You can use the QuickTime atom functions to extract atoms from that container. You can also use those functions to access the hot spot atom list. All hot spot atoms are contained in the hot spot parent atom.

Special Considerations

The node information atom container returned by this function is a copy of the atom container maintained internally by the QuickTime VR Manager. You should dispose of the node information atom container, by calling `QTDisposeAtomContainer` (II–1164), when you’re finished using it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Getting Scene and Node Information” (V–2906)

Related Java Methods

```
quicktime.std.movies.AtomContainer.fromQTVRInstanceNode(),
quicktime.vr.QTVRInstance.getNodeInfo()
```

See Also

QTVRGetNodeType

Obtains the type of a movie node.

```
OStype QTVRGetNodeType (
    QTVRInstance  qtvr,
    UInt32        nodeID );
```

qtv

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

nodeID

A node ID. Pass `kQTVRCurrentNode` for the current node.

function result The type of the node specified by the *nodeID* parameter in the QuickTime VR movie specified by the *qtv* parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Getting Scene and Node Information” (V–2906)

Related Java Methods

quicktime.vr.QTVRInstance.getNodeType()

QTVRGetPanAngle

Obtains the pan angle of a QuickTime VR movie.

```
float QTVRGetPanAngle (
    QTVRInstance  qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

function result A floating-point value that represents the current pan angle of the QuickTime VR movie specified by the qtvr parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Manipulating Viewing Angles and Zooming” (V–2905)

Related Java Methods

quicktime.vr.QTVRInstance.getPanAngle()

See Also

Use QTVRSetPanAngle (II–1431) to set the pan angle of a movie.

QTVRGetQTVRInstance

Obtains an instance of a QuickTime VR movie.

```
OSErr QTVRGetQTVRInstance (
    QTVRInstance  *qtvr,
    Track         qtvrTrack,
    MovieController mc );
```

QTVRGetQTVRInstance

qtvr

On return, an instance of the specified QuickTime VR movie.

qtvrTrack

A QTVR track contained in a QuickTime movie. You can obtain a reference to this track by calling QTVRGetQTVRTrack (II–1375).

mc

An identifier for the movie controller to be associated with the new QuickTime VR movie instance. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

function result If qtvrTrack does not specify a QTVR track, this function returns NIL in the qtvr parameter and an error code as its function result. See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

You need a QuickTime VR movie instance to call most other QuickTime VR functions. This function returns, in the qtvr parameter, an instance of the QuickTime VR movie specified by the qtvrTrack parameter. Here’s an example of code that gets a QTVR instance:

```
// QTVRGetQTVRInstance coding example
// See “Discovering QuickTime,” page 390
QTVRInstance MyGetQTVRInstanceFromMC (MovieController mc)
{
    Track          track = NIL;
    QTVRInstance    qtvrinstance = NIL;
    Movie           movie = NIL;

    //Get the movie from the movie controller.
    movie = MCGetMovie(mc);

    if (movie != NIL) {
        //Get the first QTVR track in the movie.
        track = QTVRGetQTVRTrack(movie, 1);

        //Get a QTVR instance for that QTVR track.
        if (track != NIL) {
            QTVRGetQTVRInstance(qtvrinstance, track, mc);
            //Set our units to be degrees.
            if (qtvrinstance != NIL)
                QTVRSetAngularUnits(qtvrinstance, kQTVRDegrees);
        }
    }
}
```

```

    }

    return qtvrintstance;
}

```

Special Considerations

It's not necessary to dispose of a QuickTime VR movie instance.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing QuickTime VR Movie Instances” (V–2904)

Related Java Methods

quicktime.vr.QTVRInstance.QTVRInstance()

QTVRGetQTVRTrack

Obtains a QTVR track contained in a QuickTime movie to use in the QTVRGetQTVRInstance (II–1373) call.

```

Track QTVRGetQTVRTrack (
    Movie      theMovie,
    SInt32     index );

```

theMovie

A QuickTime movie.

index

The index of the desired QTVR track.

function result A track identifier for the QTVR track that has the index specified by the index parameter in the QuickTime movie specified by the theMovie parameter. If there is no such track, or the movie is not a QTVR movie, this function returns the value NIL.

Discussion

Here's an example of using this function to help get an instance of a QTVR movie running in a movie controller:

```

// QTVRGetQTVRTrack coding example
// See “Discovering QuickTime,” page 390
QTVRInstance MyGetQTVRInstanceFromMC (MovieController mc)

```

QTVRGetQTVRTrack

```
{
    Track          track = NIL;
    QTVRInstance    qtvrinstance = NIL;
    Movie           movie = NIL;

    //Get the movie from the movie controller.
    movie = MCGetMovie(mc);

    if (movie != NIL) {
        //Get the first QTVR track in the movie.
        track = QTVRGetQTVRTrack(movie, 1);

        //Get a QTVR instance for that QTVR track.
        if (track != NIL) {
            QTVRGetQTVRInstance(qtvrinstance, track, mc);
            //Set our units to be degrees.
            if (qtvrinstance != NIL)
                QTVRSetAngularUnits(qtvrinstance, kQTVRDegrees);
        }
    }

    return qtvrinstance;
}
```

Version Notes

Introduced in QuickTime 3 or earlier. QuickTime VR 2.1 supports files with at most one QTVR track; hence the value for the `index` parameter of a movie made with QuickTime VR 2.1 is always 1. Panorama and object movies made with QuickTime VR version 1.0 have no QTVR track. This function returns the track ID of the panorama track for version 1.0 panorama movies and the track ID of the image video track for version 1.0 object movies.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing QuickTime VR Movie Instances” (V–2904)

Related Java Methods

`quicktime.std.movies.Movie.getQTVRTrack()`

See Also

Use `QTVRGetQTVRInstance` (II–1373) to get a QuickTime VR movie instance from the track identifier returned by this function.

QTVRGetTiltAngle

Obtains the tilt angle of a QuickTime VR movie.

```
float QTVRGetTiltAngle (
    QTVRInstance  qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II-1373).

function result A floating-point value that represents the current tilt angle of the QuickTime VR movie specified by the qtvr parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Manipulating Viewing Angles and Zooming” (V-2905)

Related Java Methods

quicktime.vr.QTVRInstance.getTiltAngle()

See Also

Use QTVRSetTiltAngle (II-1434) to set the tilt angle of a movie.

QTVRGetViewAnimation

Obtains the current state of view animation for an object node.

```
Boolean QTVRGetViewAnimation (
    QTVRInstance  qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II-1373).

function result A Boolean value that indicates the current state of view animation for the object node specified by the qtvr parameter. It is TRUE if view animation is enabled, FALSE otherwise.

Special Considerations

This function is valid only for object nodes.

QTVRGetViewCenter

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use QTVREnableViewAnimation (II–1342) to set the state of view animation for an object node.

QTVRGetViewCenter

Obtains the view center of a QuickTime VR movie.

```
OSErr QTVRGetViewCenter (
    QTVRInstance      qtvr,
    QTVRFloatPoint    *viewCenter );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

viewCenter

On entry, a pointer to a QTVRFloatPoint (IV–2396) structure. On return, that structure contains the current view center of the specified movie.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Manipulating Viewing Angles and Zooming” (V–2905)

Related Java Methods

quicktime.vr.QTVRInstance.getViewCenter()

See Also

Use QTVRSetViewCenter (II–1437) to set the view center of a movie.

QTVRGetViewCurrentTime

Obtains the current time in the current view.

```
TimeValue QTVRGetViewCurrentTime (
    QTVRInstance    qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

function result The current time in the current view of the object node specified by the qtvr parameter. The returned value is always greater than or equal to 0 and less than or equal to the value returned by QTVRGetCurrentViewDuration (II–1360).

Special Considerations

QTVRGetViewCurrentTime is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use QTVRSetViewCurrentTime (II–1438) to set the current time of an object node.

QTVRGetViewingLimits

Obtains the current viewing limits of a QuickTime VR movie.

```
OSErr QTVRGetViewingLimits (
    QTVRInstance    qtvr,
    UInt16          kind,
    float           *minValue,
    float           *maxValue );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

kind

The type of viewing limits to be returned (see below).

QTVRGetViewParameter

minValue

On entry, a pointer to a floating-point value. On return, the minimum viewing limit of the specified type is copied into that value.

maxValue

On entry, a pointer to a floating-point value. On return, the maximum viewing limit of the specified type is copied into that value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

kind Constants

`kQTVRPan`

The pan angle. The associated value is of type `float`.

`kQTVRTilt`

The tilt angle. The associated value is of type `float`.

`kQTVRFieldOfView`

The field of view. The associated value is of type `float`.

Discussion

This function returns, in the floating-point values pointed to by the `minValue` and `maxValue` parameters, the current minimum and maximum values for angles whose type is specified by the `kind` parameter. The maximum field of view of a panoramic node can be limited by the size of the back buffer and the current aspect ratio of the movie’s graphics world. The values returned by this function are unaffected by the current control settings.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Determining Viewing Limits and Constraints” (V–2912)

See Also

This function returns information about the physical viewing limits of a panorama or object. To get information about the current viewing constraints, use `QTVRGetConstraints` (II–1352).

QTVRGetViewParameter

Undocumented

```
OSErr QTVRGetViewParameter (
    QTVRInstance  qtvr,
```

```

    UInt32      viewParameter,
    void        *value,
    UInt32      flagsIn,
    UInt32      *flagsOut );

```

qtvr

Undocumented

viewParameter

Undocumented

value

Undocumented

flagsIn

Undocumented

flagsOut

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeVR.h

QTVRGetViewRate

Obtains the current view rate of an object node.

```

float QTVRGetViewRate (
    QTVRInstance  qtvr );

```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

function result The current view rate of the object node specified by the qtvr parameter. A view rate is a floating-point value in the range from –100.0 to +100.0. An object node’s default view rate is stored in the movie file.

Special Considerations

This function is valid only for object nodes.

QTVRGetViewState

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use QTVRSetViewRate (II–1439) to set the view rate of an object node.

QTVRGetViewState

Obtains the value of a view state.

```
OSErr QTVRGetViewState (
    QTVRInstance      qtvr,
    QTVRViewStateType viewStateType,
    UInt16             *state );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

viewStateType

A view state type (see below).

state

On entry, a pointer to a short integer. On return, that integer is set to the current value of the specified type of view state.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

viewStateType Constants

kQTVRDefault

The default view state of the current view. The default view state is a set of object images defined by view duration, row, and column. The default view state image for a given view is displayed when the mouse button is not down.

kQTVRCurrent

The current view state of the current view. The current view state can be any of the view states authored in an object movie. Setting the current view state does not change the view state designated as the *kQTVRDefault* or *kQTVRMouseDown* view state. The effect of changing the current view state lasts until a transition to the mouse-down or default view state occurs.

kQTVRMouseDown

The mouse-down state of the current view. The mouse-down view state is a set of object images defined by view duration, row, and column. The mouse-down view state image for a given view is displayed when the user holds the mouse button down while the cursor is over an object movie.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use QTVRSetViewState (II–1440) to set the value of a view state.

QTVRGetViewStateCount

Obtains the number of view states of an object node.

```
UInt16 QTVRGetViewStateCount (
    QTVRInstance    qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

function result The number of view states associated with the object node specified by the qtvr parameter. The number of view states in an object movie is defined by the movie file.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V–2909)

QTVRGetVisible

QTVRGetVisible

Obtains a movie's visibility state.

```
Boolean QTVRGetVisible (  
    QTVRInstance    qtvr );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II-1373).

function result A Boolean value that indicates whether the QuickTime VR movie specified by the qtvr parameter is visible (TRUE) or not (FALSE).

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Imaging Characteristics” (V-2910)

See Also

Use QTVRSetVisible (II-1441) to set the visibility state of a movie.

QTVRGetVisibleHotSpots

Obtains a list of the currently visible hot spots in a QuickTime VR movie.

```
UInt32 QTVRGetVisibleHotSpots (  
    QTVRInstance    qtvr,  
    Handle          hotSpots );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II-1373).

hotSpots

On entry, a valid handle to a block of memory. On return, that block of memory is filled with a list of the IDs of the visible hot spots in the specified QuickTime VR movie. If necessary, the handle is resized to hold all the hot spot IDs. Accordingly, the handle must be unlocked at the time you call this function.

function result The number of hot spot IDs returned though the hotSpots parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Hot Spots” (V–2906)

QTVRGetVRWorld

Obtains the VR world atom container for a movie.

```
OSErr QTVRGetVRWorld (
    QTVRInstance      qtvr,
    QTAtomContainer    *VRWorld );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

VRWorld

On return, a pointer to an atom container that contains information about the specified movie.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function returns, in the *VRWorld* parameter, a pointer to an atom container that contains general scene information about the QuickTime VR movie specified by the *qtvr* parameter, as well as a list of all the nodes in that movie. You can use the QuickTime atom functions to extract atoms from that container.

Special Considerations

The VR world atom container returned by this function is a copy of the atom container maintained internally by the QuickTime VR Manager. You should dispose of the VR world atom container by calling QTDisposeAtomContainer (II–1164) when you’re finished using it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Getting Scene and Node Information” (V–2906)

QTVRGoToNodeID

Related Java Methods

```
quicktime.std.movies.AtomContainer.fromQTVRInstanceWorld(),  
quicktime.vr.QTVRInstance.getVRWorld()
```

QTVRGoToNodeID

Sets the current node of a movie.

```
OSErr QTVRGoToNodeID (  
    QTVRInstance    qtvr,  
    UInt32          nodeID );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II-1373).

nodeID

The ID of the node you want to be the current node. The QuickTime VR Manager defines several constants (see below) for specific nodes.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

nodeID Constants

kQTVRCurrentNode

The current node. Moving to the current node has the effect of restoring the default view in that node.

kQTVRPreviousNode

The previous node. The QuickTime VR Manager maintains a list (which is limited only by the available memory) of the nodes visited.

kQTVRDefaultNode

The default node in the scene.

Special Considerations

Setting the current node also sets the pan, tilt, and field of view of the new current node to their default values. As a result, if you wish to set non-default angles, you should call this function before you call QTVRSetPanAngle (II-1431), QTVRSetTiltAngle (II-1434), or QTVRSetFieldOfView (II-1420).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Getting Scene and Node Information” (V-2906)

Related Java Methods

`quicktime.vr.QTVRInstance.goToNodeID()`

See Also

Use `QTVRGetCurrentNodeID` (II–1359) to get the current node ID.

QTVRInstallInterceptProc

Installs or removes an intercept procedure for a QuickTime VR Manager function.

```
OSErr QTVRInstallInterceptProc (
    QTVRInstance      qtvr,
    QTVRProcSelector  selector,
    QTVRInterceptUPP  interceptProc,
    SInt32             refCon,
    UInt32             flags );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

selector

A selector (see below) that indicates which QuickTime VR function to intercept.

interceptProc

A **Universal Procedure Pointer** for a `QTVRInterceptProc` (III–2130) callback. Set this parameter to `NIL` to remove a previously installed intercept procedure.

refCon

A reference constant to be passed to your intercept callback. Use this parameter to point to a data structure containing any information your callback needs.

flags

Unused. Set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

selector Constants

`kQTVRSetPanAngleSelector`

Intercept the `QTVRSetPanAngle` (II–1431) function.

`kQTVRSetTiltAngleSelector`

Intercept the `QTVRSetTiltAngle` (II–1434) function.

`kQTVRSetFieldOfViewSelector`

Intercept the `QTVRSetFieldOfView` (II–1420) function.

QTVRInstallInterceptProc

kQTVRSetViewCenterSelector

Intercept the QTVRSetViewCenter (II–1437) function.

kQTVRMouseEnterSelector

Intercept the QTVRMouseEnter (II–1392) function.

kQTVRMouseWithinSelector

Intercept the QTVRMouseWithin (II–1401) function.

kQTVRMouseLeaveSelector

Intercept the QTVRMouseLeave (II–1394) function.

kQTVRMouseDownSelector

Intercept the QTVRMouseDown (II–1391) function.

kQTVRMouseStillDownSelector

Intercept the QTVRMouseStillDown (II–1395) function.

kQTVRMouseUpSelector

Intercept the QTVRMouseUp (II–1398) function.

kQTVRTriggerHotSpotSelector

Intercept the QTVRTriggerHotSpot (II–1443) function.

kQTVRGetHotSpotTypeSelector

Intercept the QTVRGetHotSpotType (II–1364) function.

Discussion

This function installs the procedure specified by the `interceptProc` parameter as an intercept procedure for the QuickTime VR function specified by the `selector` parameter for the QuickTime VR movie specified by the `qtvr` parameter. Your intercept procedure is called whenever QuickTime VR is about to execute the function you are intercepting.

Your procedure can simply replace the intercepted function, by returning `TRUE` in the `cancel` parameter; or it can call through to the intercepted function, by calling `QTVRCallInterceptedProc` (II–1336); or it can allow the intercepted function to execute when the intercept procedure returns, by returning `FALSE` in the `cancel` parameter.

Here’s an example of using this function:

```
// QTVRInstallInterceptProc coding example
// See “Discovering QuickTime,” page 398
QTVRInterceptUPP MyInstallInterceptProcedure (QTVRInstance qtvrinstance)
{
    QTVRInterceptUPP    lpfnIntercept;
```

```

    lpfnIntercept = NewQTVRInterceptProc(MyInterceptProc);
    QTVRInstallInterceptProc(qtvrinstance, kQTVRSetPanAngleSelector,
                                lpfnIntercept, 0, 0);

    return lpfnIntercept
}

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Intercepting QuickTime VR Manager Routines” (V–2908)

Related Java Methods

quicktime.vr.QTVRInstance.installInterceptProc()

QTVRInteractionNudge

Translates the image and displays the new view or rotates the object in a particular direction and displays its new appearance.

```

OSErr QTVRInteractionNudge (
    QTVRInstance      qtvr,
    QTVRNudgeControl  direction );

```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

direction

The direction of the nudge (see below). The type of adjustment depends on the nudge interaction mode, which you can set with QTVRSetInteractionProperty (II–1425). If the nudge interaction mode is kQTVRNudgeRotate, the action of QTVRInteractionNudge is to rotate the object in the specified direction. If the nudge interaction mode is kQTVRNudgeTranslate, the action of QTVRInteractionNudge is to translate the image in the specified direction. If the nudge interaction mode is kQTVRNudgeSameAsMouse, the action of QTVRInteractionNudge is determined by the current mouse mode, which you can determine by calling QTVRGetCurrentMouseMode (II–1358).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

QTVRInteractionNudge

direction Constants

kQTVRRight

Nudge the current view to the right.

kQTVRUpRight

Nudge the current view up and to the right.

kQTVRUp

Nudge the current view up.

kQTVRUpLeft

Nudge the current view up and to the left.

kQTVRLeft

Nudge the current view to the left.

kkQTVRDownLeft

Nudge the current view down and to the left.

kQTVRDown

Nudge the current view down.

kQTVRDownRight

Nudge the current view down and to the right.

Discussion

This function adjusts the current view of the movie specified by the `qtvr` parameter as indicated by the `direction` parameter. The type of adjustment depends on the property setting for nudge interaction mode, which you can set with `QTVRSetInteractionProperty` (II–1425).

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Manipulating Viewing Angles and Zooming” (V–2905)

See Also

Use `QTVRGetInteractionProperty` (II–1368) and `QTVRSetInteractionProperty` (II–1425) to get and set the nudge mode and direction properties.

QTVRMouseDown

Handles the user's clicking the mouse button when the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

```
OSErr QTVRMouseDown (
    QTVRInstance    qtvr,
    Point           pt,
    UInt32          when,
    UInt16          modifiers,
    UInt32          *hotSpotID,
    WindowPtr       w );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II-1373).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the *w* parameter.

when

The time, in the number of **ticks** (sixtieths of a second) since system startup, when the mouse-down event was posted.

modifiers

A short integer, each bit of which is represented by a constant (see below) that provides information about the state of the **modifier keys** and the mouse button at the time the event was posted. More than one bit may be set.

hotSpotID

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w

A pointer to a graphics world.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

modifiers Constants

activeFlag

A window is being activated or a mouse-down event has caused a foreground switch.

btnState

The mouse button has been released.

QTVRMouseEnter

`cmdKey`

The Command key is being pressed.

`shiftKey`

The Shift key is being pressed.

`alphaLock`

The Caps Lock key is being pressed

`optionKey`

The Option key is being pressed.

`controlKey`

The Control key is being pressed.

Discussion

This function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. `QTVRMouseDown` also performs any other tasks that are typically performed when the user clicks the mouse button when the cursor is in a QuickTime VR movie.

Special Considerations

You need to call `QTVRMouseDown` only if you have disabled mouse-down tracking for the specified QuickTime VR movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Handling Events” (V-2907)

See Also

Use `QTVRSetMouseDownTracking` (II-1429) to change the mouse-down tracking state of a QuickTime VR movie. Use `QTVRMouseUp` (II-1398) and `QTVRMouseStillDown` (II-1395) to handle the mouse button’s being held down and released.

QTVRMouseEnter

Handles the user’s moving the cursor into a QuickTime VR movie for which mouse-over tracking is disabled.

```
OSErr QTVRMouseEnter (
    QTVRInstance  qtvr,
    Point         pt,
```

```
UInt32      *hotSpotID,  
WindowPtr   w );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II-1373).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

hotSpotID

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w

A pointer to a graphics world.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. `QTVRMouseEnter` also performs any other tasks that are typically performed when the user first moves the cursor into a QuickTime VR movie.

Special Considerations

You need to call `QTVRMouseEnter` only if you have disabled mouse-over tracking for the specified QuickTime VR movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Handling Events” (V-2907)

See Also

Use `QTVRSetMouseOverTracking` (II-1431) to change the mouse-over tracking state of a QuickTime VR movie. Use `QTVRMouseWithin` (II-1401) and `QTVRMouseLeave` (II-1394) to handle the cursor’s remaining in and leaving a QuickTime VR movie.

QTVRMouseLeave

QTVRMouseLeave

Handles the user's moving the cursor out of a QuickTime VR movie for which mouse-over tracking is disabled.

```
OSErr QTVRMouseLeave (  
    QTVRInstance  qtvr,  
    Point         pt,  
    WindowPtr     w );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II-1373).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the *w* parameter.

w

A pointer to a graphics world.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This function performs any tasks that are typically performed when the user moves the cursor out of a QuickTime VR movie.

Special Considerations

You need to call `QTVRMouseLeave` only if you have disabled mouse-over tracking for the specified QuickTime VR movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Handling Events” (V-2907)

See Also

Use `QTVRSetMouseOverTracking` (II-1431) to change the mouse-over tracking state of a QuickTime VR movie. Use `QTVRMouseEnter` (II-1392) and `QTVRMouseWithin` (II-1401) to handle the cursor's entering and remaining in a QuickTime VR movie.

QTVRMouseStillDown

Handles the user's holding down the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

```
OSErr QTVRMouseStillDown (
    QTVRInstance    qtvr,
    Point           pt,
    UInt32          *hotSpotID,
    WindowPtr       w );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II-1373).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the *w* parameter.

hotSpotID

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w

A pointer to a graphics world.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This function returns, in the long integer pointed to by the *hotSpotID* parameter, the ID of the hot spot in the QuickTime VR movie specified by the *qtvr* parameter that lies directly under the point specified by the *pt* parameter. If no hot spot lies under that point, the long integer is set to 0. This function also performs any other tasks that are typically performed when the user holds down the mouse button when the cursor is in a QuickTime VR movie. You should call this function repeatedly for as long as the user holds down the mouse button while the cursor is in the specified QuickTime VR movie.

Special Considerations

You need to call this function only if you have disabled mouse-down tracking for the specified QuickTime VR movie. Applications running on operating systems other than Mac OS should use the extended form of this function, `QTVRMouseStillDownExtended` (II-1396).

QTVRMouseStillDownExtended

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Handling Events” (V-2907)

See Also

Use QTVRSetMouseDownTracking (II-1429) to change the mouse-down tracking state of a QuickTime VR movie. Use QTVRMouseDown (II-1391) and QTVRMouseUp (II-1398) to handle the mouse button’s being clicked and released.

QTVRMouseStillDownExtended

Handles the user’s holding down the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

```
OSErr QTVRMouseStillDownExtended (
    QTVRInstance    qtvr,
    Point           pt,
    UInt32          *hotSpotID,
    WindowPtr       w,
    UInt32          when,
    UInt16          modifiers );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II-1373).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the w parameter.

hotSpotID

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w

A pointer to a graphics world.

when

The current time as the number of **ticks** (sixtieths of a second) since system startup.

modifiers

A short integer, each bit of which is represented by a constant (see below) that provides information about the state of the **modifier keys** and the mouse button at the time the event was posted. More than one bit may be set.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

modifiers Constants

activeFlag

A window is being activated or a mouse-down event has caused a foreground switch.

btnState

The mouse button has been released.

cmdKey

The Command key is being pressed.

shiftKey

The Shift key is being pressed.

alphaLock

The Caps Lock key is being pressed

optionKey

The Option key is being pressed.

controlKey

The Control key is being pressed.

Discussion

This function uses the same intercept as QTVRMouseStillDown (II–1395) but has two additional parameters. Applications that intercept this function should always check the paramCount field to make sure it is 5 before using the last two fields. You should call this function repeatedly for as long as the user holds down the mouse button while the cursor is in the specified QuickTime VR movie.

Special Considerations

You need to call this function only if you have disabled mouse-down tracking for the specified QuickTime VR movie. Internally, QuickTime VR always uses this function instead of QTVRMouseStillDown. Developers implementing their own mouse down tracking don’t need to use the extended version unless they also intercept the procedure and need the added parameters.

Version Notes

Introduced in QuickTime 3 or earlier.

QTVRMouseUp

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Handling Events” (V–2907)

See Also

Use QTVRSetMouseDownTracking (II–1429) to change the mouse-down tracking state of a QuickTime VR movie. Use QTVRMouseDown (II–1391) and QTVRMouseUpExtended (II–1399) to handle the mouse button’s being clicked and released.

QTVRMouseUp

Handles the user’s releasing the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

```
OSErr QTVRMouseUp (
    QTVRInstance    qtvr,
    Point           pt,
    UInt32          *hotSpotID,
    WindowPtr       w );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the *w* parameter.

hotSpotID

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w

A pointer to a graphics world.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Discussion

This function returns, in the long integer pointed to by the *hotSpotID* parameter, the ID of the hot spot in the QuickTime VR movie specified by the *qtvr* parameter that lies directly under the point specified by the *pt* parameter. If no hot spot lies under that point, the long integer is set to 0. this function also performs any other tasks that are typically performed when the user releases the mouse button after clicking it when the cursor is in a QuickTime VR movie.

Special Considerations

You need to call this function only if you have disabled mouse-down tracking for the specified QuickTime VR movie. Applications running on operating systems other than Mac OS should use the extended form of this function, `QTVRMouseUpExtended` (II–1399).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Handling Events” (V–2907)

See Also

Use `QTVRSetMouseDownTracking` (II–1429) to change the mouse-down tracking state of a QuickTime VR movie. Use `QTVRMouseDown` (II–1391) and `QTVRMouseStillDown` (II–1395) to handle the mouse button’s being clicked and held down.

QTVRMouseUpExtended

Handles the user’s releasing the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

```
OSErr QTVRMouseUpExtended (
    QTVRInstance    qtvr,
    Point           pt,
    UInt32          *hotSpotID,
    WindowPtr       w,
    UInt32          when,
    UInt16          modifiers );
```

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

`pt`

The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

`hotSpotID`

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

QTVRMouseUpExtended

w

A pointer to a graphics world.

when

The time, in the number of **ticks** (sixtieths of a second) since system startup, when the mouse-up event was posted.

modifiers

A short integer, each bit of which is represented by a constant (see below) that provides information about the state of the **modifier keys** and the mouse button at the time the event was posted. More than one bit may be set.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

modifiers Constants

activeFlag

A window is being activated or a mouse-down event has caused a foreground switch.

btnState

The mouse button has been released.

cmdKey

The Command key is being pressed.

shiftKey

The Shift key is being pressed.

alphaLock

The Caps Lock key is being pressed

optionKey

The Option key is being pressed.

controlKey

The Control key is being pressed.

Discussion

This function uses the same intercept as the `QTVRMouseUp` function but has two additional parameters. Applications that intercept this function should always check the `paramCount` field to make sure it is 5 before using the last two fields.

Special Considerations

You need to call `QTVRMouseUp` (II–1398) or this function only if you have disabled mouse-down tracking for the specified QuickTime VR movie. Internally, QuickTime VR always uses the this function function instead of `QTVRMouseUp`. Developers implementing their own mouse down tracking don’t need to use the extended version unless they also intercept the procedure and need the added parameters.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Handling Events” (V–2907)

QTVRMouseWithin

Handles the user’s leaving the cursor in a QuickTime VR movie for which mouse-over tracking is disabled.

```
OSErr QTVRMouseWithin (
    QTVRInstance  qtvr,
    Point         pt,
    UInt32        *hotSpotID,
    WindowPtr     w );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the *w* parameter.

hotSpotID

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w

A pointer to a graphics world.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You should call this function repeatedly for as long as the cursor remains in the specified QuickTime VR movie.

Special Considerations

You need to call `QTVRMouseWithin` only if you have disabled mouse-over tracking for the specified QuickTime VR movie.

Version Notes

Introduced in QuickTime 3 or earlier.

QTVRNudge

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Handling Events” (V–2907)

See Also

Use QTVRSetMouseOverTracking (II–1431) to change the mouse-over tracking state of a QuickTime VR movie. Use QTVRMouseEnter (II–1392) and QTVRMouseLeave (II–1394) to handle the cursor’s entering and leaving a QuickTime VR movie.

QTVRNudge

Turns one step in a particular direction and displays the new view.

```
OSErr QTVRNudge (
    QTVRInstance      qtvr,
    QTVRNudgeControl  direction );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

direction

The direction of the nudge (see below). Any value of the *direction* parameter that is not predefined is mapped to the closest defined value.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

direction Constants

kQTVRRight

Nudge the current view to the right.

kQTVRUpRight

Nudge the current view up and to the right.

kQTVRUp

Nudge the current view up.

kQTVRUpLeft

Nudge the current view up and to the left.

kQTVRLeft

Nudge the current view to the left.

kkQTVRDownLeft

Nudge the current view down and to the left.

kQTVRDown

Nudge the current view down.

kQTVRDownRight

Nudge the current view down and to the right.

Discussion

This function adjusts the current view of the movie specified by the `qtvr` parameter as indicated by the `direction` parameter. In particular, it turns one step in the indicated direction and displays the new view. For example, to move to the next view that is right and up from the current view, set the `direction` parameter to `kQTVRUpRight` (that is, $\pi/4$ radians, or 45 degrees). For objects, if no view is located at the adjacent object view defined by the nudge direction and wrapping is off in the desired direction, then this function remains at the current view and returns the result code `constraintReachedErr`. For objects, this function is useful for changing to an adjacent view without having to know the new pan and tilt angles. The direction of the nudge is affected by the current control settings.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Manipulating Viewing Angles and Zooming” (V-2905)

Related Java Methods

`quicktime.vr.QTVRInstance.nudge()`

See Also

Use `QTVRSetPanAngle` (II-1431) and `QTVRSetTiltAngle` (II-1434) to move to a new view specified using pan and tilt angles.



QTVRPanToColumn

Obtains the column number in the object image array that corresponds to a pan angle.

```
short QTVRPanToColumn (
    QTVRInstance  qtvr,
    float         panAngle );
```

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II-1373).

QTVRPtToAngles

panAngle

A pan angle.

function result The zero-based column number in the current object image array that corresponds to the pan angle specified by the panAngle parameter

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Converting Angles and Points” (V–2911)

Related Java Methods

quicktime.vr.QTVRInstance.panToColumn()

QTVRPtToAngles

Obtains the pan and tilt angles of a point.

```
OSErr QTVRPtToAngles (
    QTVRInstance  qtvr,
    Point          pt,
    float          *panAngle,
    float          *tiltAngle );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

pt

A point, in the local coordinates of the graphics world of the specified movie.

panAngle

On entry, a pointer to a floating-point value. On return, that value contains the pan angle of the specified point.

tiltAngle

On entry, a pointer to a floating-point value. On return, that value contains the tilt angle of the specified point.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

For a panorama, each point in the current view corresponds to a particular pan and tilt angle, with the point at the center of the view corresponding to the panorama's current pan and tilt angle. This function returns, in the floating-point values pointed to by the `panAngle` and `tiltAngle` parameters, the pan and tilt angles of the point specified by the `pt` parameter.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Converting Angles and Points” (V–2911)

Related Java Methods

```
quicktime.vr.QTVRInstance.ptToPanAngle(),
quicktime.vr.QTVRInstance.ptToTiltAngle()
```

QTVRPtToHotSpotID

Obtains the ID of the hot spot, if any, that lies beneath a point.

```
OSErr QTVRPtToHotSpotID (
    QTVRInstance  qtvr,
    Point         pt,
    UInt32        *hotSpotID );
```

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

`pt`

A point, in the local coordinates of the graphics world of the specified movie.

`hotSpotID`

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

QTVRRefreshBackBuffer

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Hot Spots” (V–2906)

QTVRRefreshBackBuffer

Refreshes the QTVR back buffer.

```
OSErr QTVRRefreshBackBuffer (
    QTVRInstance    qtvr,
    UInt32          flags );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

flags

Unused. Set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function refreshes some or all of the back buffer associated with the QuickTime VR movie specified by the *qtvr* parameter by reloading the appropriate data from the diced frames in the panorama image track. You can call this function either in a back buffer imaging procedure or elsewhere in your application. If you call this function in a back buffer imaging procedure, only the current rectangle (that is, the rectangle specified by the procedure’s *drawRect* parameter) is refreshed. If you call this function outside of a back buffer imaging procedure, all areas of interest specified in the most recent call to QTVRSetBackBufferImagingProc (II–1411) are refreshed.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Accessing Image Buffers” (V–2914)

See Also

See QTVRBackBufferImagingProc (III–2127) for information about back buffer imaging procedures.

QTVRReplaceCursor

Replaces any of the standard QuickTime VR cursors with your own custom cursor.

```
OSErr QTVRReplaceCursor (
    QTVRInstance      qtvr,
    QTVRCursorRecord  *cursRecord );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

cursRecord

A pointer to a QTVRCursorRecord (IV–2395) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function replaces one or more of the standard QuickTime VR cursors associated with the instance specified by the *qtvr* parameter with the cursors specified in the cursor record pointed to by the *cursRecord* parameter. If the type field of the specified cursor record is kQTVRUseDefaultCursor, the default cursor for the given **resource** ID is reloaded; in this case, the *handle* field of that record should be set to NIL.

Special Considerations

This function replaces the standard cursors only for the specified QuickTime VR movie instance. To replace the standard cursors for all QuickTime VR movie instances you create, you need to call this function for each such instance.

Version Notes

Introduced in QuickTime 3 or earlier. Note that QuickTime VR 2.1 makes a copy of the cursor handle specified in the cursor record. The application is responsible for disposing of its own cursor handle.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing QuickTime VR Movie Interactions” (V–2912)

QTVRRowToTilt

Obtains the tilt angle that corresponds to a row number in a QTVR object image array.

```
float QTVRRowToTilt (
```

QTVRSetAngularUnits

```
QTVRInstance    qtvr,  
short           row );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

row

A row number.

function result The tilt angle that corresponds to the zero-based row number in the object image array specified by the *row* parameter.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Converting Angles and Points” (V–2911)

Related Java Methods

`quicktime.vr.QTVRInstance.rowToTilt()`

QTVRSetAngularUnits

Sets the type of unit used when specifying QTVR angles.

```
OSErr QTVRSetAngularUnits (  
    QTVRInstance    qtvr,  
    QTVRAngularUnits units );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

units

A constant (see below) that indicates the type of angular units to use.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

units Constants

kQTVRDegrees

Angles are specified using degrees. This is the default type of angle specification.

kQTVRRadians

Angles are specified using radians

Discussion

This function sets the type of angular units to be used in all subsequent QuickTime VR Manager calls for the QuickTime VR movie specified by the `qtvr` parameter to the unit type specified by the `units` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Converting Angles and Points” (V–2911)

Related Java Methods

`quicktime.vr.QTVRInstance.setAngularUnits()`

See Also

Use `QTVRGetAngularUnits` (II–1344) to get the type of angular unit used by a QuickTime VR movie.

QTVRSetAnimationSetting

Sets the state of an animation setting for an object node.

```
OSErr QTVRSetAnimationSetting (
    QTVRInstance          qtvr,
    QTVRObjectAnimationSetting setting,
    Boolean               enable );
```

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

`setting`

An animation setting (see below).

`enable`

A Boolean value that indicates whether the specified animation setting is to be enabled for the specified object node (TRUE) or disabled (FALSE).

QTVRSetAnimationSetting

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

setting Constants

kQTVRPalindromeViewFrames

Play a back-and-forth animation of the frames of the current view. The frames of the current view play with a positive or negative frame rate; the frame rate sign is switched each time the view end time (equal to the view duration) or the view start time (always 0) is reached.

kQTVRStartFirstViewFrames

Play the frame animation starting with the first frame in the view (that is, at the view start time). This setting is useful if a sound track is associated by time with object views. Even if the object view contains no animation, setting this flag allows any sound authored to play with the view to play from the beginning. When this flag is clear, each new object view begins playing using the current view time of the previous view.

kQTVRDontLoopViewFrames

Don’t loop the frame animation. Animation frames and sound stop playing when the view duration is reached.

kQTVRPlayEveryViewFrame

Play every view frame. Animation plays all frames regardless of play rate. The play rate is used to adjust the duration in which a frame appears but no frames are skipped so the rate is not exact. When this property is set, sound tracks are not played.

kQTVRSyncViewToFrameRate

Synchronize the view animation to the frame animation. When view animation is enabled, the object views play at the same rate and animation settings as the frame animation rate and settings. This is useful if animation must be synchronized precisely across multiple views or a sound track is to be played during view animation instead of during frame animation.

kQTVRPalindromeViews

Play a back-and-forth animation of the views of the current node. When view animation is enabled, the object views play with a positive or negative view rate; the view rate sign is switched each time an object’s pan equals the object’s minimum pan limit or the object’s maximum pan limit (the first column in a row or the last column in a row, respectively).

kQTVRPlayStreamingViews

When an object movie is streaming in from a network, this flag indicates whether the frames corresponding to the row in which the default view appears are to be played as they are loaded.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Object Nodes” (V-2909)

See Also

Use QTVRGetAnimationSetting (II-1344) to get the current state of an animation setting for an object node.

QTVRSetBackBufferImagingProc

Installs or removes a QTVR back buffer imaging procedure.

```
OSErr QTVRSetBackBufferImagingProc (
    QTVRInstance      qtvr,
    QTVRBackBufferImagingUPP  backBufferImagingProc,
    UInt16            numAreas,
    QTVRAreaOfInterest  areasOfInterest[],
    SInt32            refCon );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II-1373).

backBufferImagingProc

A **Universal Procedure Pointer** for a QTVRBackBufferImagingProc (III-2127) callback. To remove a previously installed back buffer imaging procedure, pass NIL.

numAreas

The number of area of interest structures in the array pointed to by the areasOfInterest parameter.

areasOfInterest[]

A pointer to an array of QTVRAreaOfInterest (IV-2392) structures. Each structure defines a rectangular areas about which you want your back buffer imaging procedure to be notified. Your procedure is called for each area of interest as it becomes visible or not visible. You indicate when you want your procedure to be called for a particular area of interest by setting flags in the flags field in the corresponding area of interest structure. The width of the area

QTVRSetBackBufferPrefs

of interest is limited by the size of the back buffer. If the back buffer is less than the full cache size, then the area of interest can be no wider than half the size of the back buffer. (For vertical cylinder geometries, limiting factor would be the height of the buffer.) For a full cache back buffer, the width of the area of interest can be the full size of the buffer. If the width limit is exceeded, this function returns `constraintReachedErr`.

refCon

A reference constant. This value is passed to the specified back buffer imaging callback. Use this parameter to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function installs the procedure specified by the `backBufferImagingProc` parameter as a back buffer imaging procedure for the panoramic node specified by the `qtvr` parameter. You can use that procedure to draw directly into the back buffer. Coordinates in the back buffer are dependent on the current correction mode; as a result, you need to indicate the area you’re interested in drawing into by specifying a pan angle and tilt angle to determine the upper-left corner of the area and a height and width relative to that corner. Specifying a height and width instead of a second pair of pan and tilt angles for the bottom-right coordinate allows the rectangle to wrap around the edge of the panorama.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Accessing Image Buffers” (V–2914)

See Also

Use `QTVRSetPrescreenImagingCompleteProc` (II–1432) to install or remove a prescreen buffer imaging completion procedure.

QTVRSetBackBufferPrefs

Sets the resolution, pixel format, and size of the back buffer maintained internally by QuickTime VR for caching a panoramic image in a particular pixel format.

`OSErr QTVRSetBackBufferPrefs (`

```
QTVRInstance    qtvr,
UInt32          geometry,
UInt16          resolution,
UInt32          cachePixelFormat,
SInt16          cacheSize );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

geometry

The type and orientation of the panorama data (see below).

resolution

The desired image resolution (see below).

cachePixelFormat

The desired pixel format for the back buffer (see below).

cacheSize

The desired size for the panorama back buffer (see below).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

geometry Constants

kQTVRUseMovieGeometry

Value is 0.

kQTVRVerticalCylinder

Vertical cylinder geometry. Value is 'vcyl'.

resolution Constants

kQTVRDefaultRes

The default resolution of the image.

kQTVRFullRes

The full resolution of the image.

kQTVRHalfRes

One-half the full resolution of the image.

kQTVRQuarterRes

One-quarter the full resolution of the image.

cachePixelFormat Constants

k16LE555PixelFormat

16-bit LE RGB 555 (Windows)

k16BE565PixelFormat

16-bit BE RGB 565



QTVRSetBackBufferPrefs

k16LE565PixelFormat

16-bit LE RGB 565

k24BGRPixelFormat

24-bit BGR

k32ARGBPixelFormat

32-bit ARGB

k32BGRAPixelFormat

32-bit BGRA (Matrox)

k32ABGRPixelFormat

32-bit ABGR

k32RGBAPixelFormat

32-bit RGBA

cacheSize Constants

kQTVRMinimumCache

The minimum cache size required to display the specified VR movie.

kQTVRSuggestedCache

The suggested cache size, a cache large enough to allow full zooming out of the panorama.

kQTVRFullCache

The full cache size (that is, a cache that is large enough to fit the entire panorama in memory). This is the default cache size.

Discussion

This function sets the resolution, pixel format, and size of the panorama back buffer for the movie specified by the `qtv` parameter to the values specified by the `resolution` parameter and the `cachePixelFormat` and `cacheSize` parameters. You can specify a resolution that isn't contained in the movie file; if you do so, QuickTime VR takes the highest resolution image in the file and reduces it to fit into the specified buffer size. If you specify an unsupported pixel format, this function may return an error.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: "Managing VR Memory" (V-2913)

See Also

Use `QTVRGetAvailableResolutions` (II–1346) to determine which resolutions are supported by a node. Use `QTVRGetBackBufferMemInfo` (II–1347) to determine the memory requirements for the preferred settings.

QTVRSetConstraints

Sets the constraints of a VR movie.

```
OSErr QTVRSetConstraints (
    QTVRInstance  qtvr,
    UInt16        kind,
    float          minValue,
    float          maxValue );
```

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

`kind`

The type of constraint to set (see below).

`minValue`

A floating-point value that contains the desired minimum constraint of the specified type.

`maxValue`

A floating-point value that contains the desired maximum constraint of the specified type.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

kind Constants

`kQTVRPan`

The pan angle. The associated value is of type `float`.

`kQTVRTilt`

The tilt angle. The associated value is of type `float`.

`kQTVRFieldOfView`

The field of view. The associated value is of type `float`.

Discussion

This function sets the minimum and maximum constraints of the type specified by the `kind` parameter to the values specified by the `minValue` and `maxValue` parameters. Note that when you want to specify a pan angle constraint, the `minValue` and

QTVRSetControlSetting

`maxValue` parameters should be specified so that a clockwise sweep from `minValue` to `maxValue` selects the desired angular expanse. For example, to constrain panning in the 90-degree expanse that spreads out 45 degrees on each side of the pan angle 0 degrees, you should set the `minValue` parameter to 315 degrees and the `maxValue` parameter to 45 degrees. Similarly, to constrain panning in the remaining 270-degree expanse, you should set the `minValue` parameter to 45 degrees and the `maxValue` parameter to 315 degrees.

Special Considerations

The values passed to this function are unaffected by the current control settings.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Determining Viewing Limits and Constraints” (V–2912)

Related Java Methods

`quicktime.vr.QTVRInstance.setConstraints()`

See Also

Use `QTVRGetConstraints` (II–1352) to get a movie’s minimum and maximum constraints.

QTVRSetControlSetting

Sets the state of a control setting for a QTVR object node.

```
OSErr QTVRSetControlSetting (
    QTVRInstance      qtvr,
    QTVRControlSetting setting,
    Boolean            enable );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

setting

A control setting (see below).

enable

A Boolean value that indicates whether the specified control setting is to be enabled for the specified object node (TRUE) or disabled (FALSE).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

setting Constants**kQTVRWrapPan**

Set wrapping during panning. When this control setting is enabled, the user can wrap around from the current pan constraint maximum value to the pan constraint minimum value (or vice versa) using the mouse or arrow keys. In addition, calling `QTVRSetPanAngle` (II–1431) with a pan angle that is either less than or greater than the current pan constraints results in a pan angle within the current pan constraints. Similarly, `QTVRWrapAndConstrain` (II–1446) returns a pan angle within the current pan constraints, and `QTVRNudge` (II–1402) wraps views after it reaches the maximum or minimum constraint value.

When this control setting is disabled, the user cannot wrap around from the current pan constraint maximum value to the pan constraint minimum value (or vice versa) using the mouse or arrow keys. In addition, `QTVRSetPanAngle` returns the result code `constraintReachedErr` when passed a pan angle that is either less than or greater than the current pan constraints. Similarly, `QTVRWrapAndConstrain` returns a pan angle that is one of the current pan constraints, and `QTVRNudge` returns the result code `constraintReachedErr` when it reaches the maximum or minimum constraint value.

Note that a view animation stops when a constraint is reached unless palindrome view animation or wrapping during panning is enabled.

kQTVRWrapTilt

Set wrapping during tilting. When this control setting is enabled, the user can wrap around from the current tilt constraint maximum value to the tilt constraint minimum value (or vice versa) using the mouse or arrow keys. In addition, calling `QTVRSetTiltAngle` (II–1434) with a tilt angle that is either less than or greater than the current tilt constraints results in a tilt angle within the current tilt constraints. Similarly, `QTVRWrapAndConstrain` (II–1446) returns a tilt angle within the current tilt constraints, and `QTVRNudge` (II–1402) wraps views after it reaches the maximum or minimum constraint value.

When this control setting is disabled, the user cannot wrap around from the current tilt constraint maximum value to the tilt constraint minimum value (or vice versa) using the mouse or arrow keys. In addition, the `QTVRSetTiltAngle` function returns the result code `constraintReachedErr` when passed a tilt angle that is either less than or greater than the current tilt constraints. Similarly, `QTVRWrapAndConstrain` returns a tilt angle that is one of the current tilt constraints, and `QTVRNudge` returns the result code `constraintReachedErr` when it reaches the maximum or minimum constraint value.

QTVRSetControlSetting

kQTVRCanZoom

Set zooming to be active. When this control setting is enabled, the user can change the current field of view using the zoom-in and zoom-out keys on the keyboard (or using the VR controller buttons). In addition, you can use `QTVRSetFieldOfView` (II-1420) to alter the current field of view. Similarly, `QTVRWrapAndConstrain` (II-1446) returns a field of view within the current field of view constraints.

When this control is disabled, the user cannot change the current field of view using the zoom-in and zoom-out keys on the keyboard (or using the VR controller buttons). In addition, the `QTVRSetFieldOfView` function returns the result code `constraintReachedErr` and doesn't change the field of view. Similarly, `QTVRWrapAndConstrain` returns the current field of view

kQTVRReverseHControl

Reverse the direction of the horizontal control. When this control setting is enabled, the user can change an object's horizontal view using the mouse or the keyboard arrows with reversed values for mouse horizontal motion and keyboard left and right arrows. (In other words, the left arrow key causes panning to the right, and moving the mouse right causes panning to the left.) Similarly, `QTVRNudge` (II-1402) nudges left when you pass the value 0 and right when you pass the value 180. This control setting is useful when an object's frames have been authored in reverse order

kQTVRReverseVControl

Reverse the direction of the vertical control. When this control setting is enabled, the user can change an object's vertical view using the mouse or the keyboard arrows with reversed values for mouse vertical motion and keyboard up and down arrows. (In other words, the up arrow key causes tilting down, and moving the mouse down causes tilting up.) Similarly, `QTVRNudge` (II-1402) nudges up when you pass the value 270 and down when you pass the value 90. This control setting is useful when an object's frames have been authored in reverse order.

kQTVRSwapHVControl

Exchange the horizontal and vertical controls. When this setting is enabled, the user can pan left using the up arrow key and tilt up using the left arrow key. Similarly, `QTVRNudge` (II-1402) nudges up when you pass the value 180 and down when you pass the value 0. This control setting is useful if an object is a single-row or multirow movie containing vertically changing images. This is especially useful when enabling view animation for an object with animating vertical changes in view (because objects will animate only along rows, not along columns).

kQTVRTranslation

Set translation to be active. When this setting is enabled, the user can translate using the mouse when either the translation key is held down or the controller translation mode button is toggled on. In addition, you can use QTVRSetViewCenter (II–1437) to set a horizontal and vertical position in the current display coordinate system. When this setting is disabled, QTVRWrapAndConstrain (II–1446) always returns the current view center, and QTVRSetViewCenter (II–1437) returns the result code `constraintReachedErr` and doesn’t change the current view center.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use QTVRGetControlSetting (II–1355) to get the current state of a control setting for an object node.

QTVRSetEnteringNodeProc

Installs or removes a node-entering procedure.

```
OSErr QTVRSetEnteringNodeProc (
    QTVRInstance      qtvr,
    QTVREnteringNodeUPP enteringNodeProc,
    SInt32             refCon,
    UInt32             flags );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

enteringNodeProc

A **Universal Procedure Pointer** for a QTVREnteringNodeProc (III–2128) callback. To remove a previously installed QTVREnteringNodeProc callback, set `enteringNodeProc` to `NIL`.

QTVRSetFieldOfView

refCon

A reference constant. This value is passed to the specified node-entering callback. Use this parameter to point to a data structure containing any information your callback needs.

flags

Unused. Set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function installs the procedure specified by the `enteringNodeProc` parameter as a node-entering procedure for the QuickTime VR movie specified by the `qtvr` parameter. Your procedure is called whenever a node is entered (either in response to user actions or in response to QuickTime VR Manager functions that change nodes). The reference constant specified by the `refCon` parameter is passed unchanged to that node-entering procedure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing QuickTime VR Movie Interactions” (V–2912)

Related Java Methods

```
quicktime.vr.QTVRInstance.setEnteringNodeProc(),  
quicktime.vr.QTVRInstance.removeEnteringNodeProc()
```

See Also

Use `QTVRSetLeavingNodeProc` (II–1428) to install or remove a node-leaving procedure.

QTVRSetFieldOfView

Sets the vertical field of view of a QuickTime VR movie.

```
OSErr QTVRSetFieldOfView (  
    QTVRInstance  qtvr,  
    float          fieldOfView );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

fieldOfView

The desired vertical field of view for the specified movie. This value is constrained by the maximum field of view of the movie. Values that lie outside that limit are clipped to the maximum. Pan and tilt angle values are also clipped if, when combined with the current field of view, they would cause an image to lie outside the current constraints.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. If the control setting `kQTVRCanZoom` is disabled, the field of view is unchanged and this function returns the result code `constraintReachedErr`. You can use `QTVRSetControlSetting` (II–1416) to control the setting of `kQTVRCanZoom`.

Special Considerations

The pan and tilt angles are subject to the current pan and tilt range constraints, as imposed by the viewing limits and the current field of view. Accordingly, if you want to change the field of view, you should do so before adjusting the pan or tilt angles. Otherwise, the pan and tilt angles are clipped against the current field of view, which may result in an incorrect view when you alter the field of view.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Manipulating Viewing Angles and Zooming” (V–2905)

Related Java Methods

`quicktime.vr.QTVRInstance.setFieldOfView()`

See Also

Use `QTVRGetFieldOfView` (II–1360) to get the vertical field of view of a QuickTime VR movie.

QTVRSetFrameRate

Sets the frame rate of an object node.

```
OSErr QTVRSetFrameRate (
    QTVRInstance  qtvr,
    float         rate );
```

QTVRSetImagingProperty

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

rate

The desired frame rate of the specified movie. A frame rate is a floating-point value in the range from –100.0 to +100.0. Positive values indicate forward rates, and negative values indicate reverse rates. Set this parameter to 0 to stop the movie. If the value specified lies outside the valid range, this function returns the result code `constraintReachedErr` and sets the frame rate to the nearest constraint.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is most useful when an object is being viewed with a looping animation. (The current view of the object may contain frames that are played in a loop, as specified by the file format.) You can use this function to change the frame rate of the loop.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use `QTVRGetFrameRate` (II–1362) to get the frame rate of an object node.

QTVRSetImagingProperty

Sets the value of an imaging property of a movie.

```
OSErr QTVRSetImagingProperty (
    QTVRInstance      qtvr,
    QTVRImagingMode   imagingMode,
    UInt32             imagingProperty,
    SInt32             propertyValue );
```

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

`imagingMode`

An imaging mode (see below).

`imagingProperty`

An imaging property (see below).

`propertyValue`

The desired value for the specified imaging property for the specified mode.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

imagingMode Constants

`kQTVRStatic`

The panorama is static

`kQTVRMotion`

The panorama is in motion; that is, an action such as panning or zooming is occurring.

`kQTVRAAllModes`

All currently defined imaging modes. You can specify this imaging mode when calling `QTVRSetImagingProperty` (II–1422) to assign a value to a particular imaging property for all imaging modes. You can also use this imaging mode in the `imagingMode` field of a `QTVRPanoImagingAtom` (IV–2398) structure to indicate a default value for a particular imaging property for all imaging modes.

imagingProperty Constants

`kQTVRImagingCorrection`

The correction mode property (see below). This property determines the type of image correction to be applied when imaging a panoramic view.

`kQTVRImagingQuality`

The imaging quality property (see below). This property determines the quality of the output image.

`kQTVRImagingDirectDraw`

The direct-drawing property. This property determines the immediate destination of an image. The value of this property (as specified by the `propertyValue` parameter) is interpreted as a Boolean value. If the value is `TRUE`, then images are computed and drawn directly to the final destination without first being drawn in the prescreen buffer maintained by QuickTime VR. This value provides the fastest overall frame rate but may result in relatively slower individual frame drawing for high-quality, high-resolution images on lower

QTVRSetImagingProperty

performance systems. If the value of this property is `FALSE`, then images are computed and drawn first into the prescreen buffer and then to the final destination. This value provides the shortest imaging time for each individual frame but results in a reduced overall frame rate.

Note that the `kQTVRImagingDirectDraw` property cannot always be supported. During updates, QuickTime VR's warping engine might ignore a value of `TRUE` and draw the image first into the prescreen buffer and then into the final destination

`kQTVRImagingCurrentMode`

The current imaging mode property. When you pass this property selector to `QTVRGetImagingProperty` (II-1365), it returns, in the `propertyValue` parameter, the current imaging mode (that is, either `kQTVRStatic` or `kQTVRMotion`). If the `kQTVRImagingDefaultValue` bit is set, the default imaging mode is returned. You can get but not set the value of this property. You might want to determine the current imaging mode in an imaging callback procedure if what you draw depends on the current imaging mode.

`kQTVRImagingDefaultValue`

The default value specifier. You can OR this value with any of the imaging property types to get or set the default value of that property type

kQTVRImagingCorrection Constants

`kQTVRNoCorrection`

Apply no warping. The source panorama is reproduced directly, with no image correction.

`kQTVRPartialCorrection`

Apply one-dimensional (horizontal) warping. This kind of correction is often sufficient, especially for outdoor scenes.

`kQTVRFullCorrection`

Apply two-dimensional warping. This correction mode produces images in correct perspective from the source panorama.

kQTVRImagingQuality Constants

`codecMinQuality`

The minimum valid value for a CodecQ field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

Discussion

Default values for all imaging properties can be contained in a QuickTime VR movie file. If no defaults are specified in a movie file, the QuickTime VR Manager uses these values: for static mode, the `kQTVRImagingQuality` property is `codecHighQuality` and `kQTVRImagingDirectDraw` is `TRUE`; for motion mode, the `kQTVRImagingQuality` property is `codecLowQuality` and `kQTVRImagingDirectDraw` is `TRUE`. The default correction mode is `kQTVRFullCorrection` for both static and motion imaging modes. Note that it would look strange to have one correction mode for static imaging and a different correction mode for motion imaging. As a result, you should typically set the `imagingMode` parameter to `kQTVRA11Modes` when setting a property of type `kQTVRImagingCorrection`.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Imaging Characteristics” (V–2910)

Related Java Methods

`quicktime.vr.QTVRInstance.setImagingProperty()`

See Also

Use `QTVRGetImagingProperty` (II–1365) to get an imaging property.

QTVRSetInteractionProperty

Sets the value of an interaction property.

```
OSErr QTVRSetInteractionProperty (
    QTVRInstance    qtvr,
    UInt32           property,
    void             *value );
```

QTVRSetInteractionProperty

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II-1373).

property

An interaction property type (see below).

value

The desired value of the specified interaction property.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

property Constants

`kQTVRInteractionMouseClickedHysteresis`

The value parameter is interpreted as an unsigned short integer that represents the mouse-click hysteresis, the distance, in pixels, from the location of a mouse-down event to the limit within which the cursor is considered not to have moved. In other words, the cursor can move that many pixels during a mouse click and still have the event considered a click. The default mouse-click hysteresis value is 2. This property is valid for panoramas only.

`kQTVRInteractionMouseClickedTimeout`

The value parameter is interpreted as an unsigned long integer that represents the mouse-click timeout, the number of **ticks** after which a mouse click times out and is automatically switched from a hot spot selection into a pan. The default mouse-click timeout value is 30 ticks (one-half second). This property is valid for panoramas only.

`kQTVRInteractionPanTiltSpeed`

The panning and tilting speed. The value parameter is interpreted as an unsigned long integer that represents the relative speed of panning and tilting. This integer should be from 1 (the slowest speed) through 10 (the fastest speed); the default panning and tilting speed is 5. This property is valid only for panoramas.

`kQTVRInteractionZoomSpeed`

The value parameter is interpreted as an unsigned long integer that represents the zooming speed, the relative speed of zooming in and out. This integer should be from 1 (the slowest speed) through 10 (the fastest speed); the default zooming speed is 5. This property is valid for both objects and panoramas.

`kQTVRInteractionTranslateOnMouseDown`

The translate flag. The value parameter is interpreted as a Boolean value that indicates whether translate mode is enabled (true) or disabled (false). When translate mode is enabled, the user can no longer pan or tilt using the mouse;

instead, dragging the cursor causes the object to translate. The default translate flag value is false. This property is valid for objects only.

kQTVRInteractionMouseMotionScale

The value parameter is interpreted as a pointer to a floating-point number that represents the mouse-motion scale, the number of degrees or radians that an object or panorama is panned or tilted when the cursor is dragged the entire width of the object bounding box. The default value is 180.0. This property is valid for objects only.

kQTVRInteractionNudgeMode

This parameter lets you set the QuickTime VR nudge mode (see below) to either rotate, translate, or be the same as the current mouse mode.

Nudge Mode Constants

kQTVRNudgeRotate

Set the nudge mode to rotate the object.

kQTVRNudgeTranslate

Set the nudge mode to translate the image.

kQTVRNudgeSameAsMouse

Set the nudge mode to the same as the current mouse mode, which you can determine by calling `QTVRGetCurrentMouseMode` (II–1358).

Discussion

This function sets the value of the interaction property of the type specified by the property parameter for the movie specified by the `qtv` parameter to the value specified by the `value` parameter. For types that occupy 32 or fewer bits of memory, you pass the desired value itself (cast to a `void*` type) in the `value` parameter. For structures and floating-point values, you must pass a pointer to the desired value in the `value` parameter. Note that floating-point values are usually stored as 32-bit values, but compilers differ in how they pass floating-point values as parameters; as a result, this function demands that floating-point values always be passed by reference.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing QuickTime VR Movie Interactions” (V–2912)

See Also

Use `QTVRGetInteractionProperty` (II–1368) to get an interaction property.

QTVRSetLeavingNodeProc

QTVRSetLeavingNodeProc

Installs or removes a node-leaving procedure.

```
OSErr QTVRSetLeavingNodeProc (
    QTVRInstance      qtvr,
    QTVRLeavingNodeUPP leavingNodeProc,
    SInt32             refCon,
    UInt32             flags );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

leavingNodeProc

A **Universal Procedure Pointer** for a `QTVRLeavingNodeProc` (III–2130) callback. To remove a previously installed `QTVRLeavingNodeProc` callback, set `leavingNodeProc` to `NIL`.

refCon

A reference constant. This value is passed to the specified node-leaving callback. Use this parameter to point to a data structure containing any information your callback needs.

flags

Unused. Set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function installs the procedure specified by the `leavingNodeProc` parameter as a node-leaving procedure for the QuickTime VR movie specified by the `qtvr` parameter. Your procedure is called whenever a node is left (either in response to user actions or in response to QuickTime VR Manager functions that change nodes). The reference constant specified by the `refCon` parameter is passed unchanged to that node-leaving procedure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing QuickTime VR Movie Interactions” (V–2912)

Related Java Methods

```
quicktime.vr.QTVRInstance.setLeavingNodeProc(),
quicktime.vr.QTVRInstance.removeLeavingNodeProc()
```

See Also

Use `QTVRSetEnteringNodeProc` (II–1419) to install or remove a node-entering procedure.

QTVRSetMouseDownTracking

Sets the state of mouse-down tracking.

```
OSErr QTVRSetMouseDownTracking (
    QTVRInstance  qtvr,
    Boolean       enable );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

enable

A Boolean value that indicates whether QuickTime VR should handle mouse-down tracking for the specified movie (TRUE) or not (FALSE).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

By default, QuickTime VR tracks mouse clicks in a QuickTime VR movie and triggers hot spots as appropriate. If you disable mouse-down tracking (by passing FALSE in the *enable* parameter), you must call `QTVRMouseDown` (II–1391), `QTVRMouseStillDown` (II–1395), and `QTVRMouseUp` (II–1398) at the appropriate times to handle user actions.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Handling Events” (V–2907)

See Also

Use `QTVRGetMouseDownTracking` (II–1370) to get the current mouse-down tracking state of a QuickTime VR movie.

QTVRSetMouseOverHotSpotProc

Installs or removes a mouse over hot spot procedure.

QTVRSetMouseOverHotSpotProc

```
OSErr QTVRSetMouseOverHotSpotProc (
    QTVRInstance      qtvr,
    QTVRMouseOverHotSpotUPP mouseOverHotSpotProc,
    SInt32             refCon,
    UInt32             flags );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

mouseOverHotSpotProc

A **Universal Procedure Pointer** for a QTVRMouseOverHotSpotProc (III–2131) callback. To remove a previously installed QTVRMouseOverHotSpotUPP callback, set mouseOverHotSpotProc to NIL.

refCon

A reference constant. This value is passed to the specified mouse over hot spot callback. Use this parameter to point to a data structure containing any information your callback needs.

flags

Unused. Set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function installs the routine specified by the mouseOverHotSpotProc parameter as a mouse over hot spot procedure for the QuickTime VR movie specified by the qtvr parameter. Subsequent user actions (such as moving the cursor over an enabled hot spot in that movie) cause the callback routine to be executed. The reference constant specified by the refCon parameter is passed unchanged to your callback routine.

Special Considerations

Your mouse over hot spot procedure is called only for enabled hot spots.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Managing Hot Spots” (V–2906)

Related Java Methods

```
quicktime.vr.QTVRInstance.setMouseOverHotSpotProc(),
quicktime.vr.QTVRInstance.removeMouseOverHotSpotProc()
```

QTVRSetMouseOverTracking

Sets the state of mouse-over tracking.

```
OSErr QTVRSetMouseOverTracking (
    QTVRInstance    qtvr,
    Boolean          enable );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

enable

A Boolean value that indicates whether QuickTime VR should handle mouse-over tracking for the specified movie (TRUE) or not (FALSE).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

By default, QuickTime VR tracks mouse movements in a QuickTime VR movie and changes the shape of the cursor as appropriate. If you disable mouse-over tracking (by passing FALSE in the *enable* parameter), you must call QTVRMouseEnter (II–1392), QTVRMouseWithin (II–1401), and QTVRMouseLeave (II–1394) at the appropriate times to handle user actions.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Handling Events” (V–2907)

See Also

Use QTVRGetMouseOverTracking (II–1370) to get the current mouse-over tracking state of a QuickTime VR movie.

QTVRSetPanAngle

Sets the pan angle of a QuickTime VR movie.

```
OSErr QTVRSetPanAngle (
    QTVRInstance    qtvr,
    float            panAngle );
```

QTVRSetPrescreenImagingCompleteProc

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

panAngle

The desired pan angle of the specified movie. This value is constrained by the maximum and minimum pan angles of the movie. If the angle falls outside of those constraints and the control setting `kQTVRWrapPan` is disabled, the angle is set to the minimum or maximum, whichever is closer. If wrapping is enabled, the pan angle is clipped to fall within the constraints. Pan angle values are also clipped if the requested pan angle, when combined with the current tilt angle and field of view, would cause an image to lie outside the current constraints.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. This function returns the result code `constraintReachedErr` if wrapping is off and the angle is set to the minimum or maximum constraint value.

Special Considerations

The pan and tilt angles are subject to the current pan and tilt range constraints, as imposed by the viewing limits and the current field of view. Accordingly, if you want to change the field of view, you should do so before adjusting the pan or tilt angles. Otherwise, the pan and tilt angles are clipped against the current field of view, which may result in an incorrect view when you alter the field of view.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Manipulating Viewing Angles and Zooming” (V–2905)

Related Java Methods

`quicktime.vr.QTVRInstance.setPanAngle()`

See Also

Use `QTVRGetPanAngle` (II–1373) to get the pan angle of a movie. Use `QTVRGetViewingLimits` (II–1379) to get the current viewing limits of a movie. Use `QTVRSetControlSetting` (II–1416) to control the setting of `kQTVRWrapPan`.

QTVRSetPrescreenImagingCompleteProc

Installs or removes a prescreen buffer imaging completion procedure.

`OSErr QTVRSetPrescreenImagingCompleteProc (`

```
QTVRInstance      qtvr,
QTVRImagingCompleteUPP  imagingCompleteProc,
SInt32            refCon,
UInt32            flags );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

imagingCompleteProc

A **Universal Procedure Pointer** for a QTVRImagingCompleteProc (III–2129) callback. To remove a previously installed QTVRImagingCompleteProc callback, set imagingCompleteProc to NIL.

refCon

A reference constant. This value is passed to the specified prescreen buffer imaging completion callback. Use this parameter to point to a data structure containing any information your callback needs.

flags

A constant (see below) that causes a draw attempt on every idle passed to the movie controller besides being called whenever QuickTime VR finishes drawing an image into the prescreen buffer.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

flags Constants

kQTVRPreScreenEveryIdle

Causes a draw attempt on every idle passed to the movie controller.

Discussion

This function installs the procedure specified by the imagingCompleteProc parameter as a prescreen buffer imaging completion procedure for the QuickTime VR movie specified by the qtvr parameter. Your procedure is called whenever QuickTime VR finishes drawing an image into the prescreen buffer. The reference constant specified by the refCon parameter is passed unchanged to that prescreen buffer imaging completion procedure.

Special Considerations

QTVRSetPrescreenImagingCompleteProc is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Accessing Image Buffers” (V–2914)

QTVRSetTiltAngle

Related Java Methods

```
quicktime.vr.QTVRInstance.setPrescreenImagingCompleteProc(),  
quicktime.vr.QTVRInstance.removePrescreenImagingCompleteProc()
```

See Also

Use `QTVRSetBackBufferImagingProc` (II–1411) to install or remove a back buffer imaging procedure.

QTVRSetTiltAngle

Sets the tilt angle of a QuickTime VR movie.

```
OSErr QTVRSetTiltAngle (  
    QTVRInstance    qtvr,  
    float            tiltAngle );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

tiltAngle

The desired tilt angle of the specified movie. This value is constrained by the maximum and minimum tilt angles of the movie. If the angle falls outside of those constraints and the control setting `kQTVRWrapTilt` is disabled, the angle is set to the minimum or maximum, whichever is closer. If wrapping is enabled, the tilt angle is clipped to fall within the constraints. Tilt angle values are also clipped if the requested tilt angle, when combined with the current pan angle and field of view, would cause an image to lie outside the current constraints.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. This function returns the result code `constraintReachedErr` if wrapping is off and the angle is set to the minimum or maximum constraint value.

Special Considerations

The pan and tilt angles are subject to the current pan and tilt range constraints, as imposed by the viewing limits and the current field of view. Accordingly, if you want to change the field of view, you should do so before adjusting the pan or tilt angles. Otherwise, the pan and tilt angles are clipped against the current field of view, which may result in an incorrect view when you alter the field of view.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Manipulating Viewing Angles and Zooming” (V–2905)

Related Java Methods

`quicktime.vr.QTVRInstance.setTiltAngle()`

See Also

Use `QTVRGetTiltAngle` (II–1377) to get the tilt angle of a movie. Use `QTVRGetViewingLimits` (II–1379) to get the current viewing limits of a movie. Use `QTVRSetControlSetting` (II–1416) to control the setting of `kQTVRWrapTilt`.

QTVRSetTransitionProperty

Sets the value of a transition property.

```
OSErr QTVRSetTransitionProperty (
    QTVRInstance    qtvr,
    UInt32          transitionType,
    UInt32          transitionProperty,
    Sint32          transitionValue );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

transitionType

A type of transition (see below).

transitionProperty

A type of transition property (see below).

transitionValue

The desired value for the specified transition property.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

transitionType Constant

`kQTVRTransitionSwing`

A transition between two views in a single node.

transitionProperty Constants

`kQTVRTransitionSpeed`

The speed at which the transition should occur. The `transitionValue` for this parameter should be an integer from 1 (the slowest transition speed) through 10 (the fastest transition speed).

QTVRSetTransitionProperty

`kQTVRTransitionDirection`

The direction in which the transition should occur. The `transitionValue` for this parameter should be a constant (see below) that indicates the direction in which the view should swing while moving from the current view to the new view. This direction can be left or right, or it can have the value `-1`, which causes the default direction to be used. By default, a swing transition occurs along the shortest route between the two views.

kQTVRTransitionDirection Constants

`kQTVRRight`

Swing the view to the right.

`kQTVRUpRight`

Swing the view up and to the right.

`kQTVRUp`

Swing the view up.

`kQTVRUpLeft`

Swing the view up and to the left.

`kQTVRLeft`

Swing the view to the left.

`kkQTVRDownLeft`

Swing the view down and to the left.

`kQTVRDown`

Swing the view down.

`kQTVRDownRight`

Swing the view down and to the right.

Discussion

This function sets the value of the transition property whose type is specified by the `transitionType` and `transitionProperty` parameters for the movie specified by the `qtvr` parameter to the value specified by the `transitionValue` parameter. Note that calling this function simply sets a transition property's value; you must still call `QTVREnableTransition` (II-1341) to enable that transition effect.

Special Considerations

`QTVRSetTransitionProperty` is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: "Managing Imaging Characteristics" (V-2910)

See Also

Use `QTVREnableTransition` (II–1341) to enable a transition effect.

QTVRSetViewCenter

Sets the view center of a QuickTime VR movie.

```
OSErr QTVRSetViewCenter (
    QTVRInstance      qtvr,
    const QTVRFloatPoint *viewCenter );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

viewCenter

A pointer to a `QTVRFloatPoint` (IV–2396) structure that contains the desired view center of the specified movie. This point is constrained by the current field of view of the movie. The values you pass in the `QTVRFloatPoint` structure are adjusted so that the magnified area does not show anything outside the view.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. If the `kQTVRTranslation` control setting is disabled, this function returns the result code `constraintReachedErr` and doesn’t change the current view center. You can use `QTVRSetControlSetting` (II–1416) to control the setting of `kQTVRTranslation`.

Special Considerations

`QTVRSetViewCenter` is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Manipulating Viewing Angles and Zooming” (V–2905)

Related Java Methods

`quicktime.vr.QTVRInstance.setViewCenter()`

See Also

Use `QTVRGetViewCenter` (II–1378) to get the view center of a movie.

QTVRSetViewCurrentTime

QTVRSetViewCurrentTime

Sets the time in the current QTVR view.

```
OSErr QTVRSetViewCurrentTime (
    QTVRInstance  qtvr,
    TimeValue     time );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

time

The desired time in the current view. This value should be greater than or equal to 0 and less than or equal to the value returned by QTVRGetCurrentViewDuration (II–1360).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. This function returns the result code `timeNotInViewErr` if the specified time value is greater than or equal to the view duration of the specified object node; in addition, it sets the current view time to 1 less than the view duration. Similarly, this function returns the result code `timeNotInViewErr` if the specified time value is less than 0; in that case, it sets the current view time to 0.

Special Considerations

QTVRSetViewCurrentTime is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use QTVRGetViewCurrentTime (II–1379) to get the current time of an object node.

QTVRSetViewParameter

Undocumented

```
OSErr QTVRSetViewParameter (
    QTVRInstance  qtvr,
    UInt32        viewParameter,
```

```
void          *value,
UInt32       flagsIn );
```

qtvr

Undocumented

viewParameter

Undocumented

value

Undocumented

flagsIn

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeVR.h

QTVRSetViewRate

Sets the view rate of a QTVR object node.

```
OSErr QTVRSetViewRate (
    QTVRInstance  qtvr,
    float         rate );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

rate

The desired view rate of the specified movie. A view rate is a floating-point value in the range from –100.0 to +100.0. Positive values indicate forward rates, and negative values indicate reverse rates. Set this parameter to 0 to stop the movie.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function sets the view rate of the object node specified by the qtvr parameter to the rate specified by the rate parameter. A node’s view rate might be modified by

QTVRSetViewState

the current animation settings. If the value specified in the rate parameter lies outside the valid range, this function returns the result code `constraintReachedErr` and sets the view rate to the nearest constraint.

Special Considerations

`QTVRSetViewRate` is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Object Nodes” (V–2909)

See Also

For the corresponding get function, see `QTVRGetViewRate` (II–1381). Use `QTVRGetViewRate` to get the current view rate of an object node.

QTVRSetViewState

Sets the value of a QTVR view state.

```
OSErr QTVRSetViewState (
    QTVRInstance      qtvr,
    QTVRViewStateType viewStateType,
    UInt16             state );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

viewStateType

A view state type (see below).

state

The desired value of the specified type of view state.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

viewStateType Constants

`kQTVRDefault`

The default view state of the current view. The default view state is a set of object images defined by view duration, row, and column. The default view state image for a given view is displayed when the mouse button is not down.

kQTVRCurrent

The current view state of the current view. The current view state can be any of the view states authored in an object movie. Setting the current view state does not change the view state designated as the **kQTVRDefault** or **kQTVRMouseDown** view state. The effect of changing the current view state lasts until a transition to the mouse-down or default view state occurs.

kQTVRMouseDown

The mouse-down state of the current view. The mouse-down view state is a set of object images defined by view duration, row, and column. The mouse-down view state image for a given view is displayed when the user holds the mouse button down while the cursor is over an object movie.

Special Considerations

QTVRSetViewState is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Object Nodes” (V–2909)

See Also

Use **QTVRGetViewState** (II–1382) to get the value of a view state.

QTVRSetVisible

Sets a VR movie’s visibility state.

```
OSErr QTVRSetVisible (
    QTVRInstance    qtvr,
    Boolean          visible );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling **QTVRGetQTVRInstance** (II–1373).

visible

A Boolean value that indicates whether the specified movie is to be visible (TRUE) or not (FALSE).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

QTVRShowDefaultView

Discussion

Setting the visibility state to `FALSE` is useful if you want to turn off imaging a QuickTime VR movie without purging the associated data from memory. When a panoramic node's visibility state is `FALSE`, the corrected image is still drawn to the prescreen buffer. You can access the data in that buffer by calling `QTVRSetPrescreenImagingCompleteProc` (II-1432).

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Imaging Characteristics” (V-2910)

See Also

Use `QTVRGetVisible` (II-1384) to get the visibility state of a movie.

QTVRShowDefaultView

Displays the default view of a QTVR node.

```
OSErr QTVRShowDefaultView (  
    QTVRInstance  qtvr );
```

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II-1373).

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

This function sets the default values of the pan angle, tilt angle, field of view, view center (for object nodes), default state, mouse-down state, and all applicable animation and control settings for the VR movie specified by the `qtvr` parameter. A VR movie's default view values are stored in the movie file.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Manipulating Viewing Angles and Zooming” (V-2905)

Related Java Methods

```
quicktime.vr.QTVRInstance.showDefaultView()
```

QTVRTiltToRow

Obtains the row number in the QTVR object image array that corresponds to a tilt angle.

```
short QTVRTiltToRow (
    QTVRInstance  qtvr,
    float         tiltAngle );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

tiltAngle

A tilt angle.

function result The zero-based row number in the current object image array that corresponds to the tilt angle specified by the `tiltAngle` parameter.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Converting Angles and Points” (V–2911)

Related Java Methods

```
quicktime.vr.QTVRInstance.tiltToRow()
```

QTVRTriggerHotSpot

Triggers a QTVR hot spot.

```
OSErr QTVRTriggerHotSpot (
    QTVRInstance  qtvr,
    UInt32        hotSpotID,
    QTAtomContainer nodeInfo,
    QTAtom        selectedAtom );
```

QTVRTriggerHotSpot

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

hotSpotID

A hot spot ID.

nodeInfo

A node information atom container, obtained from a previous call to `QTVRGetNodeInfo` (II–1371). You can pass the value 0 in this parameter to have QuickTime determine the appropriate node information atom container.

selectedAtom

The atom of the hot spot to trigger. You can pass the value 0 in this parameter to have QuickTime determine the appropriate hot spot atom.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

One way you can use this function is to execute any hot spot without the user’s having clicked it. Usually, you need only specify the *qtvr* instance and the hot spot ID. You can pass zero for the *nodeInfo* and *selectedAtom* parameters.

The more common use of this function is not in calling it directly, but in setting up an intercept procedure on it. This function is called internally by QuickTime whenever a user clicks a hot spot. You can intercept calls to trigger your custom hot spots, which allows you to perform any custom actions you desire.

When this function is called internally (and then intercepted by your intercept procedure), the *nodeInfo* and *selectedAtom* parameters have been properly set by QuickTime and are available for your use. For undefined hot spots that do not have an associated hot spot atom in the *node info* atom container, the *selectedAtom* parameter will be set to zero.

Special Considerations

You can call this function even on hot spots that are currently disabled.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Hot Spots” (V–2906)

Related Java Methods

`quicktime.vr.QTVRInstance.triggerHotSpot()`

See Also

Use `QTVRInstallInterceptProc` (II–1387) to install an intercept procedure.

QTVRUpdate

Forces an immediate update of a QuickTime VR movie image.

```
OSErr QTVRUpdate (
    QTVRInstance      qtvr,
    QTVRImagingMode   imagingMode );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

imagingMode

An imaging mode. You can specify `kQTVRCurrentMode` (see below) to use the current imaging mode.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

imagingMode Constants

`kQTVRCurrentMode`

The current imaging mode.

Discussion

This function immediately updates the image for the QuickTime VR movie specified by the *qtvr* parameter, without waiting for the next call to `MoviesTask` (II–973) in your application’s main event loop. If you plan to call this function repeatedly for a movie instance, then for improved performance you should bracket those calls with calls to `QTVRBeginUpdateStream` (II–1335) and `QTVREndUpdateStream` (II–1343).

Special Considerations

If you call this function after calling `QTVRBeginUpdateStream` (II–1335) but before calling `QTVREndUpdateStream` (II–1343), the *imagingMode* parameter passed to it must be the same as the *imagingMode* parameter passed to `QTVRBeginUpdateStream`. If you do not specify the same imaging mode to those two functions, an error will result.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Managing Imaging Characteristics” (V–2910)

QTVRWrapAndConstrain

Related Java Methods

`quicktime.vr.QTVRInstance.update()`

QTVRWrapAndConstrain

Preflights a change in the viewing or control characteristics of a QTVR object or panoramic node.

```
OSErr QTVRWrapAndConstrain (
    QTVRInstance  qtvr,
    short         kind,
    float         value,
    float         *result );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

kind

A constraint type (see below).

value

The desired value of the specified viewing characteristic.

result

On return, the value to which the specified viewing characteristic would be set if it were changed.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

kind Constants

`kQTVRPan`

The pan angle. The associated value is of type `float`.

`kQTVRTilt`

The tilt angle. The associated value is of type `float`.

`kQTVRFieldOfView`

The field of view. The associated value is of type `float`.

`kQTVRViewCenterH`

The horizontal view center. This value is valid only for this function, which returns the horizontal view center constrained in the current field of view. The minimum and maximum values of the horizontal view center change for every field-of-view setting. If the field of view of the object is the maximum field of view, the object’s horizontal view center is constrained to a single value that is

the horizontal center of the object image in display coordinates (rounding down any fractional pixels).

`kQTVRViewCenterV`

The vertical view center. This value is valid only for this function, which returns the vertical view center constrained in the current field of view. The minimum and maximum values of the vertical view center change for every field-of-view setting. If the field of view of the object is the maximum field of view, the object's vertical view center is constrained to a single value that is the vertical center of the object image in display coordinates (rounding down any fractional pixels).

Discussion

This function returns, in the `result` parameter, the constrained or wrapped value that would result from setting the viewing or control characteristic specified by the `kind` parameter to the value specified by the `value` parameter. For example, if the `kind` parameter is set to `kQTVRPan`, then this function returns the value that would result from calling `QTVRSetPanAngle` (II-1431) with its `panAngle` parameter set to the `value` parameter. Similarly, you can use this function to find the current bounds of the view center. It takes into account the current constraints and wrapping modes of the node specified by the `qtvr` parameter. This function does not change the current view or other settings of the specified object or panorama.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeVR.h`

Programming summary: “Converting Angles and Points” (V-2911)

Related Java Methods

`quicktime.vr.QTVRInstance.wrapAndConstrain()`

QuadToQuadMatrix

Undocumented

```
OSErr QuadToQuadMatrix (
    const Fixed      *source,
    const Fixed      *dest,
    MatrixRecord      *map );
```

source

Undocumented

QuadToQuadMatrix

dest

Undocumented

map

A pointer to a MatrixRecord (IV–2304) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: ImageCompression.h

Related Java Methods

quicktime.std.image.Matrix.Matrix()

Volume III

Functions R–Z and Callbacks

R

RectMatrix

Creates a matrix that performs the translate and scale operation described by the relationship between two rectangles.

```
void RectMatrix (
    MatrixRecord    *matrix,
    const Rect      *srcRect,
    const Rect      *dstRect );
```

matrix

A pointer to a `MatrixRecord` (IV–2304) structure. This function updates the contents of this matrix so that the matrix describes a transformation from points in the rectangle specified by the `srcRect` parameter to points in the rectangle specified by the `dstRect` parameter. The previous contents of the matrix are ignored.

srcRect

A pointer to the source `Rect` (IV–2399) structure.

dstRect

A pointer to the destination `Rect` (IV–2399) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Matrices” (V–2764)

Related Java Methods

`quicktime.std.image.Matrix.rect()`

RegisterComponent

Makes a component that is resident in memory available for use by applications or other clients.

```
Component RegisterComponent (
    ComponentDescription    *cd,
    ComponentRoutineUPP     componentEntryPoint,
```

RegisterComponent

```
short      global,  
Handle     componentName,  
Handle     componentInfo,  
Handle     componentIcon );
```

cd

A `ComponentDescription` (IV–2212) structure that describes the component to be registered. You must correctly fill in the fields of this record before calling this function. When applications search for components using `FindNextComponent` (I–360), the Component Manager compares the attributes you specify here with those specified by the application. If the attributes match, this function returns the component identifier.

componentEntryPoint

A **Universal Procedure Pointer** to the main entry point of the component you are registering. The routine referred to by this parameter receives all requests for the component.

global

Flags (see below) that control the scope of component registration.

componentName

A handle to the component's name. Set this parameter to `NIL` if you do not want to assign a name to the component.

componentInfo

A handle to the component's information string. Set this parameter to `NIL` if you do not want to assign an information string to the component.

componentIcon

A handle to the component's 32-by-32 pixel black-and-white icon. Set this parameter to `NIL` if you do not want to supply an icon for this component. Note that this icon is not used by QuickTime; you supply an icon only so that other components or applications can display your component's icon if needed.

function result The component identifier assigned to the component by the Component Manager. If it cannot register the component, this function returns `NIL`. If this function fails to register a component because one with identical characteristics already exists, it returns 0.

global Constants

registerCmpGlobal

Indicates that this component should be made available to other applications and clients as well as the one performing the registration. If you do not specify this flag, the component is available for use only by the registering application or component.

`registerCmpNoDuplicates`

Indicates that if a component with identical characteristics to the one being registered already exists, then the new one should not be registered. The function returns 0 in this situation. If you do not specify this flag, the component is registered even if a component with identical characteristics to the one being registered already exists.

`registerCompAfter`

Indicates that this component should be registered after all other components with the same component type. Usually components are registered before others with identical descriptions; specifying this flag overrides that behavior.

Discussion

This function registers the specified component, recording the information specified in the `cd`, `componentName`, `componentInfo`, and `componentIcon` parameters. Once a component has been registered, applications can find and open it using the standard Component Manager routines.

Special Considerations

Components you register with the this function must be in memory when you call it. Note that a component residing in your application heap remains registered until your application unregisters it or quits. A component residing in the system heap and registered by your application remains registered until your application unregisters it or until the computer is shut down.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V-2782)

See Also

If you want to register a component that is stored in a file, use `RegisterComponentResource` (III-1453).

RegisterComponentResource

Makes a component that is stored in a file available for use by applications or other clients.

```
Component RegisterComponentResource (
    ComponentResourceHandle  cr,
    short                    global );
```

RegisterComponentResource

cr

A handle to a component **resource** that describes the component to be registered. Components you register with this function must be stored in a file as a ComponentResource (IV-2219) structure. The component resource contains all the information required to register the component.

global

Flags (see below) that control the scope of component registration.

function result The component identifier assigned to the component by the Component Manager. If it cannot register the component, this function returns NIL. If this function fails to register a component because one with identical characteristics already exists, it returns 0.

global Constants

registerCmpGlobal

Indicates that this component should be made available to other applications and clients as well as the one performing the registration. If you do not specify this flag, the component is available for use only by the registering application or component.

registerCmpNoDuplicates

Indicates that if a component with identical characteristics to the one being registered already exists, then the new one should not be registered. The function returns 0 in this situation. If you do not specify this flag, the component is registered even if a component with identical characteristics to the one being registered already exists.

registerCompAfter

Indicates that this component should be registered after all other components with the same component type. Usually components are registered before others with identical descriptions; specifying this flag overrides that behavior.

Discussion

This function does not actually load the code specified by the component **resource** into memory. Rather, the Component Manager loads the component code the first time an application opens the component. If the code is not in the same file as the component resource or if the Component Manager cannot find the file, the open request fails.

Special Considerations

Note that a component registered locally by your application remains registered until your application unregisters it or quits. A component registered globally by your application remains registered until your application unregisters it or until the computer is shut down.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V-2782)

See Also

If you want to register a component that is in memory, use `RegisterComponent` (III-1451).

R**RegisterComponentResourceFile**

Registers all component **resources** in a given resource file.

```
long RegisterComponentResourceFile (
    short    resRefNum,
    short    global );
```

`resRefNum`

The reference number of the **resource** file containing the components to register.

`global`

Flags (see below) that control the scope of component registration.

function result If `RegisterComponentResourceFile` successfully registers all components in the specified **resource** file, it returns the number of components registered. If it could not register one or more of the components in the resource file or if the specified file reference number is invalid, it returns a negative function result.

global Constants

`registerCmpGlobal`

Indicates that this component should be made available to other applications and clients as well as the one performing the registration. If you do not specify this flag, the component is available for use only by the registering application or component.

`registerCmpNoDuplicates`

Indicates that if a component with identical characteristics to the one being registered already exists, then the new one should not be registered. The function returns 0 in this situation. If you do not specify this flag, the component is registered even if a component with identical characteristics to the one being registered already exists.

RemoveCallBackFromTimeBase

`registerCompAfter`

Indicates that this component should be registered after all other components with the same component type. Usually components are registered before others with identical descriptions; specifying this flag overrides that behavior.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

See Also

See `RegisterComponentResource` (III–1453).

RemoveCallBackFromTimeBase

Removes a **callback event** from the list of scheduled callback events.

```
OSErr RemoveCallBackFromTimeBase (
    QTCallBack    cb );
```

`cb`

The **callback event** for the operation. Your clock component obtains this value from the parameters passed to your `ClockCallMeWhen` (I–91) function.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Your clock component should call this function when your `ClockCancelCallBack` (I–93) function determines that your component can cancel the **callback event**.

Special Considerations

Your component should call this function only for **callback events** that were successfully added to the schedule with `AddCallBackToTimeBase` (I–21).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Toolbox Clock Support Functions” (V–2869)

RemoveImageDescriptionExtension

Removes a specified extension from an ImageDescription (IV–2285) structure.

```
OSErr RemoveImageDescriptionExtension (
    ImageDescriptionHandle desc,
    long idType,
    long index );
```

desc

A handle to an ImageDescription (IV–2285) structure.

idType

The type of extension to remove.

index

The index of the extension to remove. This is a number between 1 and the count returned by CountImageDescriptionExtensionType (I–143).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function allows an application to remove a specified extension from an ImageDescription (IV–2285) structure. Note that any extensions that are present in the structure after the deleted extension will have their index numbers renumbered.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Working With Image Descriptions” (V–2790)

RemoveMovieExecuteWiredActionsProc

Removes a MovieExecuteWiredActionsProc (III–2101) callback from a movie.

```
OSErr RemoveMovieExecuteWiredActionsProc (
    Movie theMovie,
    MovieExecuteWiredActionsUPP proc,
    void *refCon );
```

RemoveMovieResource

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

proc

A `MovieExecuteWiredActionsProc` (III–2101) callback that was previously installed using `AddMovieExecuteWiredActionsProc` (I–36).

refCon

A reference constant that is passed to your callback. Use this parameter to point to a data structure containing any information your callback needs.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

RemoveMovieResource

Removes a movie **resource** from a specified movie file.

```
OSErr RemoveMovieResource (
    short    resRefNum,
    short    resId );
```

resRefNum

Identifies the movie file that contains the movie **resource**. Your application obtains this value from `OpenMovieFile` (II–1133).

resId

ID of the **resource** to be removed.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Saving Movies” (V–2715)

Related Java Methods

`quicktime.std.movies.Movie.removeResource()`

RemoveSoundDescriptionExtension

Removes an extension from a `SoundDescription` (IV–2449) structure.

```
OSErr RemoveSoundDescriptionExtension (
    SoundDescriptionHandle desc,
    OSType                 idType );
```

desc

A handle to the `SoundDescription` (IV–2449) structure to remove the extension from.

idType

A four-byte signature identifying the type of data being removed from the `SoundDescription` structure.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.media.SoundDescription.removeExtension()`

RemoveUserData

Removes an item from a **user data list**.

```
OSErr RemoveUserData (
    UserData    theUserData,
    OSType      udType,
    long        index );
```

RemoveUserDataText

theUserData

The **user data list** for this operation. You obtain this list reference by calling GetMovieUserData (I–499), GetTrackUserData (I–546), or GetMediaUserData (I–456).

udType

The item’s type value.

index

The item’s index value. This parameter must specify an item in the **user data list** identified by the theUserData parameter.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

After the Movie Toolbox removes the item, it rennumbers the remaining items of that type so that their index values are sequential and start at 1.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

quicktime.std.movies.media.UserData.removeData()

RemoveUserDataText

Removes language-tagged text from an item in a **user data list**.

```
OSErr RemoveUserDataText (
    UserData    theUserData,
    OSType      udType,
    long        index,
    short       itlRegionTag );
```

theUserData

The **user data list** for this operation. You obtain this list reference by calling the GetMovieUserData (I–499), GetTrackUserData (I–546), or GetMediaUserData (I–456).

udType

The item's type value.

index

The item's index value. This parameter must specify an item in the **user data list** identified by the `theUserData` parameter.

itlRegionTag

The language code of the text to be removed. See “Localization Codes” (IV–2673).

function result

You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

`quicktime.std.movies.media.UserData.removeText()`

ReplaceDSequenceImageDescription

Undocumented

```
OSErr ReplaceDSequenceImageDescription (
    ImageSequence      seqID,
    ImageDescriptionHandle newDesc );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

newDesc

A handle to an `ImageDescription` (IV–2285) structure.

function result

See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `ImageCompression.h`

ResolveComponentAlias

ResolveComponentAlias

Undocumented

```
Component ResolveComponentAlias (  
    Component    aComponent );
```

aComponent

A component instance.

function result A component instance.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Components.h

RotateMatrix

Modifies the contents of a matrix so that it defines a rotation operation.

```
void RotateMatrix (  
    MatrixRecord    *m,  
    Fixed           degrees,  
    Fixed           aboutX,  
    Fixed           aboutY );
```

m

A pointer to a MatrixRecord (IV–2304) structure.

degrees

The number of degrees of rotation.

aboutX

The x coordinate of the anchor point of rotation.

aboutY

The y coordinate of the anchor point of rotation.

Discussion

This function updates the contents of a matrix so that the matrix describes a rotation operation; that is, it concatenates the rotation transformations onto whatever was initially in the matrix structure. You specify the direction and amount of rotation with the degrees parameter. You specify the point of rotation with the aboutX and aboutY parameters.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Matrices” (V–2764)

Related Java Methods

`quicktime.std.image.Matrix.rotate()`

R

RTPMPDoUserDialog

Obtains media-specific settings from the user through a dialog box.

```
ComponentResult RTPMPDoUserDialog (
    RTPMediaPacketizer    rtpm,
    ModalFilterUPP        inFilterUPP,
    Boolean                *canceled );
```

rtpm

The component instance of the media packetizer.

inFilterUPP

A `ModalFilterProc` (III–2098) callback, which may be used in a call to the Mac OS `ModalDialog` function.

canceled

On return, a `Boolean` which is `TRUE` if the user pressed the cancel button in the dialog box. If this parameter is returned `TRUE`, the settings prior to calling this function should be retained.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function invokes a media packetizer’s modal dialog to obtain user settings. If the packetizer supports “more settings,” you can put up a dialog allowing the user to enter media-specific settings. You can determine whether a packetizer has this characteristic by calling `RTPMPHasCharacteristic` (III–1471)). The settings can be obtained for storage by calling `RTPMPGetSettingsIntoAtomContainerAtAtom` (III–1469), and can be restored or set directly from an application by calling `RTPMPSetSettingsFromAtomContainerAtAtom` (III–1481).

Special Considerations

This function may be called at any time.

RTPMPFlush

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPMPFlush

Renamed RTPMPReset (III–1474).

```
ComponentResult RTPMPFlush (
    RTPMediaPacketizer    rtpm,
    SInt32                 inFlags,
    SInt32                 *outFlags );
```

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPMPGetInfo

Obtains information of various types from a media packetizer.

```
ComponentResult RTPMPGetInfo (
    RTPMediaPacketizer    rtpm,
    OSType                 inSelector,
    void                   *ioParams );
```

rtpm

The component instance of the media packetizer you want information from.

inSelector

The selector for the type information you want (see below).

ioParams

A pointer to a data structure of the appropriate type to hold the information you are requesting. You need to allocate and dispose of this data structure.

function result See “Error Codes” (IV–2663). Returns `qtsBadSelectorErr` if `inSelector` requests a selector you do not support. Returns `noErr` if there is no error.

inSelector Constants**kRTPMPPayloadTypeInfo**

The function fills a data structure of type `RTPMPPayloadTypeParams` (IV–2407), which describes the payload being used. If the `flags` field of this structure contains `kRTPMPPayloadTypeStaticFlag`, then the `payloadNumber` field contains the RTP payload number; if the `flags` field of the structure contains `kRTPMPPayloadTypeDynamicFlag`, then the `payloadName` field contains the name of the dynamic payload encoding being use. The caller must copy the `payloadName` field data for it to be persistent.

kRTPMPRTPTimeScaleInfo

The function returns the RTP time scale used by this packetizer if the time scale must be a specific value; otherwise it returns an error for this selector.

kRTPMPRequiredSampleDescriptionInfo

The function returns a handle to a `SampleDescription` (IV–2414) structure specifying that only data with the given sample description is supported; if no such structure exists, it returns an error for this selector.

kRTPMPMinPayloadSize

The function returns the minimum payload size of packets allowed by this packetizer as a value of type `UInt32`.

kRTPMPMinPacketDuration

The function returns the minimum packet duration allowed by this packetizer, in milliseconds, as a value of type `UInt32`.

kRTPMPSuggestedRepeatPktCountInfo

The function returns the suggested number of repeat packets to send for this media, as a value of type `UInt32`. This will typically be 0 for audio or video, 1 for text or MIDI.

kRTPMPSuggestedRepeatedPktSpacingInfo

The function returns the suggested temporal distance between repeat packets, in milliseconds, as a value of type `UInt32`.

Discussion

This function can be called at any time.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

For the corresponding set function, see `RTPMPSetInfo` (III–1475).

RTPMPGetMaxPacketDuration

RTPMPGetMaxPacketDuration

Reads the maximum packet duration currently set for this packetizer.

```
ComponentResult RTPMPGetMaxPacketDuration (  
    RTPMediaPacketizer    rtpm,  
    UInt32                *outMaxPacketDuration );
```

rtpm

The component instance of the media packetizer.

outMaxPacketDuration

On return, a pointer to a 32-bit integer containing the maximum packet duration, in milliseconds, that the packetizer is set to use.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The maximum allowable packet duration can change during a presentation, so you should obtain this value immediately before using it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding set function, see RTPMPSetMaxPacketDuration (III–1477).

RTPMPGetMaxPacketSize

Returns the maximum packet size, in bytes, that the packetizer is set to create.

```
ComponentResult RTPMPGetMaxPacketSize (  
    RTPMediaPacketizer    rtpm,  
    UInt32                *outMaxPacketSize );
```

rtpm

The component instance of the media packetizer.

outMaxPacketSize

On return, a pointer to a 32-bit integer containing the maximum packet size, in bytes, that the packetizer is set to create.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The maximum allowable packet size can change during a presentation, so you should obtain this value immediately before using it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

You can set the maximum packet size by calling `RTPMPSetMaxPacketSize` (III–1477).

RTPMPPGetMediaType

Obtains the data type being handled by a media packetizer.

```
ComponentResult RTPMPPGetMediaType (
    RTPMediaPacketizer    rtpm,
    OSType                *outMediaType );
```

rtpm

The component instance of the media packetizer.

outMediaType

On return, a pointer to the media's data type, such as `VideoMediaType` or `SoundMediaType`; see “Data References” (IV–2661).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

The media's data type must be set prior to calling `RTPMPSetSampleData` (III–1480). It cannot change afterward.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

You can set the media's data type by calling `RTPMPSetMediaType` (III–1478).

RTPMPPGetPacketBuilder

RTPMPPGetPacketBuilder

Obtains the component instance of the packet builder component being used by a media packetizer.

```
ComponentResult RTPMPPGetPacketBuilder (
    RTPMediaPacketizer    rtpm,
    ComponentInstance     *outPacketBuilder );
```

rtpm

The component instance of the media packetizer whose packet builder you are interested in.

outPacketBuilder

On return, a pointer to the component instance of the packet builder component in use by this media packetizer.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: *QTStreamingComponents.h*

See Also

You can set a packet builder instance by calling *RTPMPPSetPacketBuilder* (III–1479).

RTPMPPGetSettingsAsText

Return the media-specific settings of a media packetizer as text in a format presentable to the user.

```
ComponentResult RTPMPPGetSettingsAsText (
    RTPMediaPacketizer    rtpm,
    Handle                *text );
```

rtpm

The component instance of a media packetizer.

text

Return a handle to a copy of your user settings in text format. The text is formatted as simple array of characters. There is no size byte or null termination. Allocate the handle to fit the text precisely.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function expects you to return your user settings as text. It should be called only if the media packetizer supports packetizer-specific settings. To determine if your media packetizer supports this function, the application may call RTPMPHasCharacteristic (III–1471).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

Packetizer-specific settings can be set by RTPMPSetSettingsFromAtomContainerAtAtom (III–1481). You may be asked to bring up a dialog which allows the user to change the settings through RTPMPDoUserDialog (III–1463).

RTPMPGetSettingsIntoAtomContainerAtAtom

Obtains the media-specific setting of a media packetizer.

```
ComponentResult RTPMPGetSettingsIntoAtomContainerAtAtom (
    RTPMediaPacketizer    rtpm,
    QTAtomContainer        inOutContainer,
    QTAtom                 inParentAtom );
```

rtpm

The component instance of the media packetizer.

inOutContainer

The atom container that holds the settings atom, which the caller must allocate.

inParentAtom

The atom that will hold the settings.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function should be called only if the media packetizer supports packetizer-specific settings. To determine if a media packetizer supports this function, call RTPMPHasCharacteristic (III–1471).

Version Notes

Introduced in QuickTime 4.

RTPMPGetTimeBase

Programming Info

C interface file: QTStreamingComponents.h

See Also

You can set packetizer-specific settings by calling RTPMPSetSettingsFromAtomContainerAtAtom (III–1481). To bring up a dialog which the user can use to change the settings, call RTPMPDoUserDialog (III–1463).

RTPMPGetTimeBase

Returns the time base passed to a media packetizer by RTPMPSetTimeBase (III–1481).

```
ComponentResult RTPMPGetTimeBase (  
    RTPMediaPacketizer    rtpm,  
    TimeBase              *outTimeBase );
```

rtpm

The component instance of your media packetizer.

outTimeBase

A pointer to the time base passed to you by RTPMPSetTimeBase (III–1481).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding set function, see RTPMPSetTimeBase (III–1481).

RTPMPGetTimeScale

Obtains the time scale in use by a media packetizer.

```
ComponentResult RTPMPGetTimeScale (  
    RTPMediaPacketizer    rtpm,  
    TimeScale             *outTimeScale );
```

rtpm

The component instance of media packetizer component.

outTimeScale

On return, contains a pointer to the time scale in use by the packetizer. The time scale indicates the number of time units that pass in one second when the media is playing at a rate of 1.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

For the corresponding set function, see `RTPMPSetTimeScale` (III–1482).

RTPMPHasCharacteristic

Determines whether a media packetizer has a particular characteristic, such as whether it supports a user settings dialog.

```
ComponentResult RTPMPHasCharacteristic (
    RTPMediaPacketizer    rtpm,
    OSType                inSelector,
    Boolean                *outHasIt );
```

rtpm

The component instance of the media packetizer.

inSelector

A selector for the characteristic you want to know about.

outHasIt

On return, contains a Boolean value that is `TRUE` if the media packetizer has this characteristic, `FALSE` otherwise.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inSelector Constants

`kRTPMPNoSampleDataRequiredCharacteristic`

The media packetizer does not require the actual sample data to perform packetization; the caller can pass in `NIL` data pointers instead of pointers to the actual media data. See `RTPMPSetSampleData` (III–1480).

RTPMPIIdle

kRTPMPHasUserSettingsDialogCharacteristic

The media packetizer supports RTPMPDoUserDialog (III–1463), RTPMPGetSettingsIntoAtomContainerAtAtom (III–1469), and RTPMPSetSettingsFromAtomContainerAtAtom (III–1481).

kRTPMPPrefersReliableTransportCharacteristic

The media packetizer prefers its data to be sent reliably, such as in text or music tracks.

kRTPMPRequiresOutOfBandDimensionsCharacteristic

The original visual dimensions of the media data cannot be determined simply by looking at the media packets; they must be transmitted by some other method.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPMPIIdle

Called periodically in your event loop to allocate time to each media packetizer.

```
ComponentResult RTPMPIIdle (
    RTPMediaPacketizer    rtpm,
    SInt32                 inFlags,
    SInt32                 *outFlags );
```

rtpm

The component instance of the media packetizer.

inFlags

There are currently no defined flags.

outFlags

On return, contains a pointer to a signed 32-bit integer that holds a flag (see below) from the packetizer.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

outFlags Constant

kRTPMPStillProcessingData

If this flag is set, there is still data in the queue to be processed. Call this function repeatedly until this flag is not returned.

Discussion

The packetizer will use this time to process the data in its buffer. If the data has not all been processed, this function returns the `kRTPMPStillProcessingData` flag. Data is placed in the buffer by `RTPMPSetSampleData` (III–1480).

Special Considerations

The packetizer may make calls to the packet builder in response to this call.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPMPLInitialize

Initializes a media packetizer component.

```
ComponentResult RTPMPLInitialize (
    RTPMediaPacketizer    rtpm,
    SInt32                inFlags );
```

rtpm

The component instance of the media packetizer.

inFlags

A signed 32-bit integer containing the flags (see below) you wish to pass to the packetizer at start-up.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inFlags Constant

`kRTPMPRealtimeModeFlag`

The packetizer is being called in a real-time situation (for example, live broadcasting).

Discussion

The calling component must call this function before sending any data to a media packetizer or making any `RTPMPSet` calls. The calling component then calls `RTPMPSetSampleData` (III–1480) and `RTPMPIdle` (III–1472) repeatedly. The calling component passes sample data (obtained, for example, from `GetMediaSample` (I–443)), to the media packetizer by calling `RTPMPSetSampleData`. If `RTPMPSetSampleData` or `RTPMPIdle` return the flag `kRTPMPStillProcessingData`, then the calling component should call `RTPMPIdle`; if not, it is free to call `RTPMPSetSampleData` again.

RTPMPPreflightMedia

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPMPPreflightMedia

Determines whether your packetizer can work with a given media type and sample description.

```
ComponentResult RTPMPPreflightMedia (
    RTPMediaPacketizer      rtpm,
    OSType                  inMediaType,
    SampleDescriptionHandle  inSampleDescription );
```

rtpm

The component instance of your media packetizer.

inMediaType

The media type, such as 'vide'; see “Data References” (IV–2661).

inSampleDescription

A handle to the SampleDescription (IV–2414) structure.

function result Return noErr if you can packetize this type of data; return qtsUnsupportedFeatureErr if you cannot. See “Error Codes” (IV–2663).

Discussion

This function must be implemented by your packetizer. It will be called before you are asked to packetize any data.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPMPReset

Allows a media packetizer to stop packetizing its current input, set its state to idle, and flush its input buffer.

```
ComponentResult RTPMPReset (
```



```
RTPMediaPacketizer    rtpm,
SInt32                inFlags );
```

rtpm

The component instance of the media packetizer.

inFlags

A signed 32-bit integer containing any flags you are passing to the media packetizer. There are currently no defined flags.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You can use this function to stop the media packetizer and flush its input buffer when you wish to stop transmitting immediately, when you are skipping forward or backward in the stream, or if the network data connection is interrupted.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPMPSetInfo

Sets any one of several parameters for a media packetizer.

```
ComponentResult RTPMPSetInfo (
    RTPMediaPacketizer    rtpm,
    OSType                inSelector,
    const void             *ioParams );
```

rtpm

The component instance of the media packetizer.

inSelector

A selector (see below) for the type of information you wish to set.

ioParams

A pointer to a data structure of the appropriate type for the information you are passing.

function result Return `qtsBadSelectorErr` if you do not support the selector. Return `noErr` if there is no error. See “Error Codes” (IV–2663).

RTPMPSetInfo

inSelector Constants

kQTSSourceTrackIDInfo

The source track ID as a value of type UInt32.

kQTSSourceLayerInfo

The source layer number as a value of type UInt16.

kQTSSourceLanguageInfo

The source language as a value of type UInt16; see “Localization Codes” (IV–2673).

kQTSSourceTrackFlagsInfo

The source track flags as a value of type SInt32.

kQTSSourceDimensionsInfo

The source dimensions as a QTSDimensionParams (IV–2368) structure.

kQTSSourceVolumesInfo

The source audio volumes as a QTSVolumesParams (IV–2390) structure.

kQTSSourceMatrixInfo

The source matrix as a MatrixRecord (IV–2304) structure.

kQTSSourceClipRectInfo

The source clipping rectangle as a Rect (IV–2399) structure.

kQTSSourceGraphicsModeInfo

The source graphics mode as a QTSGraphicsModeParams (IV–2372) structure.

kQTSSourceBoundingRectInfo

The source bounding rectangle as a Rect (IV–2399) structure.

kQTSSourceScaleInfo

The source scale as a Point (IV–2339) structure.

kQTSSourceUserDataInfo

The source user data as a UserDataRecord (IV–2496) structure.

kQTSSourceInputMapInfo

The source input map as a QT atom container.

Discussion

This function is used to pass track-level information about the media track to be packetized, such as its track ID, layer, and transformation matrix. Return `qtsBadSelectorErr` unless your packetizer is able to transmit this kind of data to your reassembler for use in the client movie.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

For the corresponding get function, see `RTPMPGetInfo` (III–1464).

RTPMPSetMaxPacketDuration

Sets the maximum packet duration that the media packetizer is to use.

```
ComponentResult RTPMPSetMaxPacketDuration (
    RTPMediaPacketizer    rtpm,
    UInt32                inMaxPacketDuration );
```

rtpm

The component instance of the media packetizer.

inMaxPacketDuration

An unsigned 32-bit integer containing the maximum packet duration in milliseconds. This value should not be smaller than the value returned from `RTPMPGetInfo` (III–1464) with the `kRTPMPMinPacketDuration` selector.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The maximum packet duration cannot be changed during a presentation, and this function cannot be called after calling `RTPMPSetSampleData` (III–1480).

Special Considerations

If `RTPMPSetMaxPacketDuration` is not called, a default value will be used.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

For the corresponding get function, see `RTPMPGetMaxPacketDuration` (III–1466).

RTPMPSetMaxPacketSize

Sets the maximum packet size for packets created by a media packetizer.

```
ComponentResult RTPMPSetMaxPacketSize (
```

RTPMPSetMediaType

```
RTPMediaPacketizer    rtpm,  
UInt32                inMaxPacketSize );
```

rtpm

The component instance of the media packetizer.

inMaxPacketSize

An unsigned 32-bit integer specifying the maximum size, in bytes, of packets to be created. This value must not be smaller than the value returned from RTPMPGetInfo (III–1464) with the kRTPMPMinPayloadSize selector. The media packetizer will not create packets larger than this value. The limit applies only to the payload data.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The maximum packet size cannot change during a presentation. Streaming will be most efficient if this value is set to the largest packet size that can traverse the network without being split. RTPMPSetMaxPacketSize may not be called after calling RTPMPSetSampleData (III–1480).

Special Considerations

If RTPMPSetMaxPacketSize is not called, a default value will be used.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding get function, see RTPMPGetMaxPacketSize (III–1466).

RTPMPSetMediaType

Sets the type of media that a media packetizer will process.

```
ComponentResult RTPMPSetMediaType (  
    RTPMediaPacketizer    rtpm,  
    OSType                inMediaType );
```

rtpm

The component instance of the media packetizer.

inMediaType

The media type; see “Data References” (IV–2661).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The media type must be set prior to calling RTPMPSetsSampleData (III–1480) and cannot change after such calls.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding get function, see RTPMPGetMediaType (III–1467).

RTPMPSetsPacketBuilder

Selects which packet builder a media packetizer will use.

```
ComponentResult RTPMPSetsPacketBuilder (
    RTPMediaPacketizer    rtpm,
    ComponentInstance      inPacketBuilder );
```

rtpm

The component instance of the media packetizer.

inPacketBuilder

The component instance of the packet builder component to use.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

A media packetizer always sends its output to a packet builder. The specified packet builder may assemble actual RTP packets, or it may use information about the packet to build a hint track. You must set the packet builder using this call prior to any calls to RTPMPSetsSampleData (III–1480). You can also use this function to dynamically change the packet builder a media packetizer uses.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

Use RTPMPGetPacketBuilder (III–1468) to get the packet builder currently being used.

RTPMPSetSampleData

RTPMPSetSampleData

Provides sample data directly to a media packetizer component.

```
ComponentResult RTPMPSetSampleData (
    RTPMediaPacketizer      rtpm,
    const RTPMPSampleDataParams *inSampleData,
    SInt32                  *outFlags );
```

rtpm

The component instance of the media packetizer.

inSampleData

A pointer to a RTPMPSampleDataParams (IV–2407) structure containing the sample data you are passing. Calling this routine adds data cumulatively to any previous calls to this function. The data can contain any number of samples (1 or more), or a partial sample.

outFlags

Flags (see below) that indicate processing status. This function will return kRTPMPWantsMoreDataFlag if it has completed processing of all pending data. Otherwise, you must make calls to RTPMPIdle (III–1472) until this function no longer returns kRTPMPStillProcessingData.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

outFlags Constants

kRTPMPWantsMoreDataFlag

Returned if this function has completed processing of all pending data.

kRTPMPStillProcessingData

Returned if this function is still processing data.

Discussion

This routine is called to pass media data directly to a media packetizer. The packetizer will not copy this data; it will call the release callback when it is finished with it. The media packetizer may or may not make calls to the packet builder in response to this call.

Special Considerations

This call is normally followed by a series of calls to RTPMPIdle (III–1472), which grants time to the media packetizer in order to process the data passed by this function.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPMPSetsSettingsFromAtomContainerAtAtom

Sets the media-specific settings of a media packetizer, using an atom inside an atom container.

```
ComponentResult RTPMPSetsSettingsFromAtomContainerAtAtom (
    RTPMediaPacketizer    rtpm,
    QTAtomContainer        inContainer,
    QTAtom                 inParentAtom );
```

rtpm

The component instance of the media packetizer.

inContainer

The atom container that holds the settings atom.

inParentAtom

The atom that holds the settings.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function should be called only if the media packetizer supports packetizer-specific settings. To determine if a media packetizer supports this function, call RTPMPHasCharacteristic (III–1471).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

You can get packetizer-specific settings with RTPMPGetSettingsIntoAtomContainerAtAtom (III–1469). To bring up a dialog which the user can use to change the settings, call RTPMPDoUserDialog (III–1463).

RTPMPSetTimeBase

Tells your packetizer what time base is in use by the calling application.

```
ComponentResult RTPMPSetTimeBase (
```

RTPMPSetTimeScale

```
RTPMediaPacketizer    rtpm,  
TimeBase              inTimeBase );
```

rtpm

Component instance of your packetizer.

inTimeBase

The time base in use for this stream. You can query this time base to find out the current time in the stream.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function may be called during setup for a live transmission.

Special Considerations

Your packetizer should not rely on receiving this call.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

For the corresponding get function, see `RTPMPGetTimeBase` (III–1470).

RTPMPSetTimeScale

Sets the time scale the media packetizer will use.

```
ComponentResult RTPMPSetTimeScale (  
    RTPMediaPacketizer    rtpm,  
    TimeScale              inTimeScale );
```

rtpm

The component instance of the media packetizer.

inTimeScale

The time scale to use.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The time scale is the number of time units that pass in one second when the media is playing at a rate of 1. This time scale gives meaning to the times used when calling `RTPMPSetSampleData` (III–1480).

Special Considerations

The time scale must be set before calling RTPMPSetSampleData (III–1480).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding get function, see RTPMPGetTimeScale (III–1470).

RTPPBAddPacketLiteralData

Passes literal data directly to a packet builder component.

```
ComponentResult RTPPBAddPacketLiteralData (
    RTPPacketBuilder      rtpb,
    SInt32                inFlags,
    RTPPacketGroupRef     inPacketGroup,
    RTPPacketRef          inPacket,
    UInt8                 *inData,
    UInt32                inDataLength,
    RTPPacketRepeatedDataRef *outDataRef );
```

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

The packet group containing the packet into which the data will be placed. This is normally a reference returned by RTPPBBeginPacketGroup (III–1490).

inPacket

The RTP packet into which the data will be placed. This is normally a reference returned by RTPPBBeginPacket (III–1489).

inData

A pointer to the data you are passing.

inDataLength

An unsigned 32-bit integer containing the length, in bytes, of the data you are passing.

RTPPBAddPacketRepeatedData

`outDataRef`

On return, contains a pointer to a data reference. Use this reference if you wish to later tell the packet builder to use this same data again, without having to literally pass the data again. Pass in `NIL` if you do not need the packet builder to repeat the data. If you do not pass in `NIL`, you must dispose of the data explicitly by calling `RTPPBReleaseRepeatedData` (III–1497).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function will return a reference which can be used to specify the same data repeatedly without having to pass in the data again. This is done by calling `RTPPBAddPacketRepeatedData` (III–1484) with the reference which was returned by this function. For example, you can use this function to insert static header information into a packet prior to inserting media sample data. It will return a data reference you can use to insert the same static information into later packets.

Special Considerations

To specify media data to be placed in a packet, a media packetizer should call `RTPPBAddPacketSampleData` (III–1485).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

When a reference returned by `RTPPBAddPacketLiteralData` is no longer needed, it should be disposed of by using the call `RTPPBReleaseRepeatedData` (III–1497).

RTPPBAddPacketRepeatedData

Tells a packet builder component to insert previously-specified data into a packet.

```
ComponentResult RTPPBAddPacketRepeatedData (
    RTPPacketBuilder      rtpb,
    SInt32                inFlags,
    RTPPacketGroupRef     inPacketGroup,
    RTPPacketRef          inPacket,
    RTPPacketRepeatedDataRef inDataRef );
```

`rtpb`

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

The packet group containing the packet into which the data will be placed. This is normally a reference returned by RTPPBBeginPacketGroup (III–1490).

inPacket

The RTP packet into which the data will be placed. This is normally a reference returned by RTPPBBeginPacket (III–1489).

inDataRef

A reference to the data to repeat. This is normally a data reference returned by RTPPBAddPacketLiteralData (III–1483) or RTPPBAddPacketSampleData (III–1485).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Use this function to cause a packet builder component to repeatedly insert the same data into packets without having to pass the data each time. This is typically done to repeat static header information into a series of packets, or to insert previously-sent sample data into a redundant packet. The data is first specified by a call to RTPPBAddPacketLiteralData (III–1483) or RTPPBAddPacketSampleData (III–1485), which inserts the data the first time and returns a data reference. The data reference is then used with this function to send the data again.

Special Considerations

When you are done sending the repeated data, release the data structure by calling RTPPBReleaseRepeatedData (III–1497).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPPBAddPacketSampleData

Commands a packet builder component to insert media sample data into a packet.

```
ComponentResult RTPPBAddPacketSampleData (
    RTPPacketBuilder      rtpb,
    SInt32                inFlags,
    RTPPacketGroupRef     inPacketGroup,
    RTPPacketRef          inPacket,
```

RTPPBAddPacketSampleData

```
RTPMPSampleDataParams    *inSampleDataParams,  
UInt32                   inSampleOffset,  
UInt32                   inSampleDataLength,  
RTPPacketRepeatedDataRef *outDataRef );
```

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

The packet group containing the packet into which the data will be placed. This is normally a reference returned by RTPPBBeginPacketGroup (III–1490).

inPacket

The RTP packet into which the data will be placed. This is normally a reference returned by RTPPBBeginPacket (III–1489).

inSampleDataParams

A pointer to a RTPMPSampleDataParams (IV–2407) structure for the sample data you are inserting.

inSampleOffset

A 32-bit unsigned integer containing the offset into the sample media, in bytes.

inSampleDataLength

A 32-bit unsigned integer specifying the number of bytes of media sample data to insert into the packet.

outDataRef

On return, contains a pointer to a data reference. Use this reference if you wish to later tell the packet builder to use this same sample data again, without having to literally pass the data again. Pass in `NIL` if you do not need the packet builder to repeat the data. If you do not pass in `NIL`, you must dispose of the data explicitly by calling RTPPBReleaseRepeatedData (III–1497).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function will return a reference which can be used to specify the same data repeatedly without having to pass in the data again. The media packetizer specifies the offset into the media and the length of the sample to insert. You can insert data repeatedly by calling RTPPBAddPacketRepeatedData (III–1484) with the reference which was returned by RTPPBAddPacketLiteralData (III–1483).

Special Considerations

When a reference is no longer needed, it should be disposed of by using the call `RTPPBReleaseRepeatedData` (III–1497).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

R

RTPPBAddPacketSampleData64

Provides a 64-bit version of `RTPPBAddPacketSampleData` (III–1485) for large sample media.

```
ComponentResult RTPPBAddPacketSampleData64 (
    RTPPacketBuilder      rtpb,
    SInt32                inFlags,
    RTPPacketGroupRef     inPacketGroup,
    RTPPacketRef          inPacket,
    RTPMPSampleDataParams *inSampleDataParams,
    const UInt64          *inSampleOffset,
    UInt32                inSampleDataLength,
    RTPPacketRepeatedDataRef *outDataRef );
```

`rtpb`

The component instance of the packet builder component.

`inFlags`

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

`inPacketGroup`

The packet group containing the packet into which the data will be placed. This is normally a reference returned by `RTPPBBeginPacketGroup` (III–1490).

`inPacket`

The RTP packet into which the data will be placed. This is normally a reference returned by `RTPPBBeginPacket` (III–1489).

`inSampleDataParams`

A pointer to a `RTPMPSampleDataParams` (IV–2407) structure for the sample data you are inserting.

`inSampleOffset`

A 64-bit unsigned integer containing the offset into the sample media, in bytes.

RTPPBAddRepeatPacket

`inSampleDataLength`

A 32-bit unsigned integer specifying the number of bytes of media sample data to insert into the packet.

`outDataRef`

On return, contains a pointer to a data reference. Use this reference if you wish to later tell the packet builder to use this same sample data again, without having to literally pass the data again. Pass in `NIL` if you do not need the packet builder to repeat the data. If you do not pass in `NIL`, you must dispose of the data explicitly by calling `RTPPBReleaseRepeatedData` (III–1497).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

For details of using this function, see `RTPPBAddPacketSampleData` (III–1485).

RTPPBAddRepeatPacket

Undocumented

```
ComponentResult RTPPBAddRepeatPacket (
    RTPPacketBuilder    rtpb,
    SInt32              inFlags,
    RTPPacketGroupRef   inPacketGroup,
    RTPPacketRef        inPacket,
    TimeValue           inTransmissionOffset,
    UInt32              inSequenceNumber );
```

`rtpb`

The component instance of the packet builder component.

`inFlags`

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

`inPacketGroup`

The packet group containing the packet into which the data will be placed. This is normally a reference returned by `RTPPBBeginPacketGroup` (III–1490).

inPacket

The RTP packet into which the data will be placed. This is normally a reference returned by RTPPBBeginPacket (III–1489).

inTransmissionOffset

Undocumented

inSequenceNumber

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

RTPPBBeginPacket

Tells a packet builder to create a new packet.

```
ComponentResult RTPPBBeginPacket (
    RTPPacketBuilder    rtpb,
    SInt32              inFlags,
    RTPPacketGroupRef   inPacketGroup,
    UInt32              inPacketMediaDataLength,
    RTPPacketRef        *outPacket );
```

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

The packet group containing the new packet. This is normally a reference returned by RTPPBBeginPacketGroup (III–1490).

inPacketMediaDataLength

An unsigned 32-bit integer specifying the maximum length of data that will be inserted into this packet. This includes the data for all subsequent RTPPBAddPacketLiteralData (III–1483), RTPPBAddPacketSampleData (III–1485), and RTPPBAddPacketRepeatedData (III–1484) calls until the packet is closed. The

RTPPBBeginPacketGroup

value of this parameter may be larger, but must not be smaller, than the amount of data inserted in the packet.

outPacket

On return, contains a pointer to the packet. Use this reference to insert data into the packet.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The media packetizer uses this function to create each new packet, before inserting any literal, repeated, or sample data. A call to RTPPBBeginPacketGroup (III–1490) must be made before creating the first packet in a group. Data can be inserted into the packet using RTPPBAddPacketLiteralData (III–1483), RTPPBAddPacketRepeatedData (III–1484), or RTPPBAddPacketSampleData (III–1485). When the packet is complete, call RTPPPEndPacket (III–1491).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPPBBeginPacketGroup

Tells a packet builder to create a new packet group.

```
ComponentResult RTPPBBeginPacketGroup (
    RTPPacketBuilder    rtpb,
    SInt32              inFlags,
    UInt32              inTimeStamp,
    RTPPacketGroupRef   *outPacketGroup );
```

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inTimeStamp

A unsigned 32-bit integer containing the time stamp for this packet group.

outPacketGroup

On return, contains a pointer to a reference to the packet group. Use this data reference when creating a new packet or inserting data into a packet that belongs to this group.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

A media packetizer creates a packet group using this function. The data reference returned by this function is then used to create a series of packets that belong to this group. The data reference is also required when inserting data into packets.

Special Considerations

When the packet group is complete, call RTPPBEndPacketGroup (III–1492).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPPBEndPacket

Tells a packet builder that a packet is complete.

```
ComponentResult RTPPBEndPacket (
    RTPPacketBuilder    rtpb,
    SInt32              inFlags,
    RTPPacketGroupRef   inPacketGroup,
    RTPPacketRef        inPacket,
    UInt32              inTimeOffset,
    UInt32              inDuration );
```

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

The packet group containing the new packet. This is normally a reference returned by RTPPBBeginPacketGroup (III–1490).

RTPPEndPacketGroup

inPacket

The RTP packet containing the data. This is normally a reference returned by `RTPPBeginPacket` (III–1489).

inTimeOffset

The time offset at which the media sample data contained in this packet begins, in milliseconds. This offset is added to the RTP transmission time to determine when to send the packet.

inDuration

The duration of this packet, specified in milliseconds.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Call this function once when each packet is complete.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPPEndPacketGroup

Tells a packet builder component that a packet group is complete.

```
ComponentResult RTPPEndPacketGroup (
    RTPPacketBuilder    rtpb,
    SInt32              inFlags,
    RTPPacketGroupRef    inPacketGroup );
```

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

A data reference to the packet group being ended. This is normally a data reference returned by `RTPPBeginPacketGroup` (III–1490).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function should be called when all the packets in a group are complete and the media packetizer is ready either to create a new packet group or to terminate the stream.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPPBGetCallback

Gets the callback used to communicate with the caller of a media packetizer.

```
ComponentResult RTPPBGetCallback (
    RTPPacketBuilder    rtpb,
    RTPPBCallbackUPP    *outCallback,
    void                **outRefCon );
```

rtpb

The component instance of the packet builder component.

outCallback

A pointer to an RTPPBCallbackProc (III–2133) callback.

outRefCon

A handle to any data your callback needs.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding set function, see RTPPBSetCallback (III–1497).

RTPPBGetInfo

Gets information about a streaming packet builder.

```
ComponentResult RTPPBGetInfo (
    RTPPacketBuilder    rtpb,
```

RTPPBGetPacketSequenceNumber

```
OSType          inSelector,  
void            *ioParams );
```

rtpb

The component instance of the packet builder component.

inSelector

A constant (see below) that defines the type of information to retrieve.

ioParams

A pointer to the retrieved information.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inSelector Constant

kRTPBufferDelayInfo

A UInt32 value that represents the buffer delay in the current time scale.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding set function, see RTPPBSetInfo (III–1498).

RTPPBGetPacketSequenceNumber

Gets the relative sequence number for a streaming packet.

```
ComponentResult RTPPBGetPacketSequenceNumber (  
    RTPPacketBuilder    rtpb,  
    SInt32              inFlags,  
    RTPPacketGroupRef   inPacketGroup,  
    RTPPacketRef        inPacket,  
    UInt32               *outSequenceNumber );
```

rtpb

The component instance of the packet builder component.

inFlags

Undocumented

inPacketGroup

A data reference to a packet group. This is normally a data reference returned by RTPPBBeginPacketGroup (III–1490).

inPacket

The RTP packet. This is normally a reference returned by RTPPBBeginPacket (III–1489).

outSequenceNumber

A pointer to the sequence number.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding set function, see RTPPBSetPacketSequenceNumber (III–1499).

RTPPBGetPacketTimeStampOffset

Undocumented

```
ComponentResult RTPPBGetPacketTimeStampOffset (
    RTPPacketBuilder    rtpb,
    SInt32              inFlags,
    RTPPacketGroupRef   inPacketGroup,
    RTPPacketRef        inPacket,
    SInt32              *outTimeStampOffset );
```

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

The packet group containing the packet of interest. This is normally a reference returned by RTPPBBeginPacketGroup (III–1490).

inPacket

The RTP packet of interest. This is normally a reference returned by RTPPBBeginPacket (III–1489).

outTimeStampOffset

Undocumented

RTPPBGetSampleData

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

RTPPBGetSampleData

Undocumented

```
ComponentResult RTPPBGetSampleData (
    RTPPacketBuilder      rtpb,
    RTPMPSampleDataParams *inParams,
    const UInt64          *inStartOffset,
    UInt8                 *outDataBuffer,
    UInt32                 inBytesToRead,
    UInt32                 *outBytesRead,
    SInt32                 *outFlags );
```

rtpb

The component instance of the packet builder component.

inParams

A pointer to a RTPMPSampleDataParams (IV–2407) structure.

inStartOffset

Undocumented

outDataBuffer

Undocumented

inBytesToRead

Undocumented

outBytesRead

Undocumented

outFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

RTPPBReleaseRepeatedData

Lets a packet builder deallocate data that will no longer be used.

```
ComponentResult RTPPBReleaseRepeatedData (
    RTPPacketBuilder      rtpb,
    RTPPacketRepeatedDataRef  inDataRef );
```

rtpb

The component instance of the packet builder component.

inDataRef

The data reference to the repeated data. This is normally a data reference returned by RTPPBAddPacketLiteralData (III–1483) or RTPPBAddPacketSampleData (III–1485).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

You must release the data if you have allowed RTPPBAddPacketLiteralData (III–1483) or RTPPBAddPacketSampleData (III–1485) to return a data reference, even if you have not called RTPPBAddPacketRepeatedData (III–1484) must either pass NIL to the data reference when adding literal or sample data, or you must release the data by calling this function.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPPBSetCallback

Sets the callback used to communicate with the caller of a media packetizer.

```
ComponentResult RTPPBSetCallback (
    RTPPacketBuilder      rtpb,
    RTPPBCallbackUPP      inCallback,
    void                  *inRefCon );
```

rtpb

The component instance of the packet builder component.

RTPPBSetInfo

inCallback

A **Universal Procedure Pointer** that references an RTPPBCallbackProc (III–2133) callback.

inRefCon

A pointer to any data your callback needs.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding get function, see RTPPBGetCallback (III–1493).

RTPPBSetInfo

Sets information for a streaming packet builder.

```
ComponentResult RTPPBSetInfo (  
    RTPPacketBuilder    rtpb,  
    OSType               inSelector,  
    void                 *ioParams );
```

rtpb

The component instance of the packet builder component.

inSelector

A constant (see below) that defines the type of information to set.

ioParams

A pointer to the information.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inSelector Constant

kRTPBufferDelayInfo

A UInt32 value that represents the buffer delay in the current time scale.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding get function, see RTPPBGetInfo (III–1493).

RTPPBSetPacketSequenceNumber

Sets the relative sequence number for a streaming packet.

```
ComponentResult RTPPBSetPacketSequenceNumber (
    RTPPacketBuilder    rtpb,
    SInt32              inFlags,
    RTPPacketGroupRef   inPacketGroup,
    RTPPacketRef        inPacket,
    UInt32              inSequenceNumber );
```

rtpb

The component instance of the packet builder component.

inFlags

Undocumented

inPacketGroup

A data reference to a packet group. This is normally a data reference returned by RTPPBBeginPacketGroup (III–1490).

inPacket

The RTP packet. This is normally a reference returned by RTPPBBeginPacket (III–1489).

inSequenceNumber

The sequence number to be set.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding get function, see RTPPBGetPacketSequenceNumber (III–1494).

RTPPBSetPacketTimeStampOffset

Undocumented

RTPRssmAdjustPacketParams

```
ComponentResult RTPPBSetPacketTimeStampOffset (
    RTPPacketBuilder    rtpb,
    SInt32              inFlags,
    RTPPacketGroupRef   inPacketGroup,
    RTPPacketRef        inPacket,
    SInt32              inTimeStampOffset );
```

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

The packet group containing the packet of interest. This is normally a reference returned by RTPPBBeginPacketGroup (III–1490).

inPacket

The RTP packet of interest. This is normally a reference returned by RTPPBBeginPacket (III–1489).

inTimeStampOffset

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmAdjustPacketParams

Called by the base reassembler when it is processing a packet, allowing your packet reassembler to adjust the packet parameters before the packet is processed.

```
ComponentResult RTPRssmAdjustPacketParams (
    RTPReassembler      rtp,
    RTPRssmPacket       *inPacket,
    SInt32              inFlags );
```

rtp

The component instance of your packet reassembler

inPacket

A pointer to the packet whose parameters can be adjusted.

inFlags

A signed 32-bit integer containing any flags (see below) being passed to your packet reassembler.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inFlags Constants

Undocumented

Undocumented

Discussion

Your packet reassembler can adjust the following parameters in each packet: `payloadHeaderLength`, `dataLength`, `serverEditParams`, and `chunkFlags`. If your packet reassembler does not implement this function, or takes no action, the default for these parameters will be: `payloadHeaderLength` = fixed header length that is set (default is 0); `dataLength` = `<packet data>` – `transportHeaderLength` – `payloadHeaderLength`; no `serverEditParams`; `chunkFlags` = 0.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPRssmClearCachedPackets

Forces the base reassembler to flush all packets currently queued in its lists.

```
ComponentResult RTPRssmClearCachedPackets (
    RTPReassembler    rtpr,
    SInt32             inFlags );
```

rtpr

The component instance of the base reassembler.

inFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

RTPRssmComputeChunkSize

Discussion

This function retains the last sequence number and related information. It is useful only when the base reassembler is operating with the `kRTPRssmQueueAndUseMarkerBitFlag` flag set; see `RTPRssmSetCapabilities` (III–1520).

Version Notes

Introduced in QuickTime 4.1. Replaces `RTPRssmFlushPackets`.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

See `RTPRssmReset` (III–1516).

RTPRssmComputeChunkSize

Lets your packet reassembler compute the size of a chunk, based on the packet list for the chunk, using your own algorithm.

```
ComponentResult RTPRssmComputeChunkSize (
    RTPReassembler    rtptr,
    RTPRssmPacket      *inPacketListHead,
    SInt32              inFlags,
    UInt32              *outChunkDataSize );
```

rtptr

The component instance of your packet reassembler.

inPacketListHead

A pointer to the list of packets that make up this chunk.

inFlags

A signed 32-bit integer containing any flags being passed to your packet reassembler.

outChunkDataSize

You should return a pointer to an unsigned 32-bit variable containing the calculated size for this chunk.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is called once for each packet list. Implement this function only if you need to override the base reassembler’s default computation. If you do not implement this call, the base reassembler will compute the chunk size by summing the data lengths for all packets in the list.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmCopyDataToChunk

Lets your packet reassembler write the chunk data, based on the list of packets for the chunk, using your own algorithm.

```
ComponentResult RTPRssmCopyDataToChunk (
    RTPReassembler    rtpr,
    RTPRssmPacket      *inPacketListHead,
    UInt32             inMaxChunkDataSize,
    SHChunkRecord      *inChunk,
    SInt32             inFlags );
```

rtpr

The component instance of your packet reassembler.

inPacketListHead

A pointer to the list of packets that make up this chunk.

inMaxChunkDataSize

An unsigned 32-bit integer containing the maximum allowable chunks size.

inChunk

A pointer to the chunk record. Write the chunk data to this record.

inFlags

A 32-bit signed integer containing any flags (see below) being passed to your media packetizer.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inFlags Constants

Undocumented

Undocumented

Discussion

This function is useful, for example, when an H.261 packet reassembler must adjust the byte at packet boundaries. Implement this function only if you need to override the base reassembler’s default behavior. If you do not implement this function, the base reassembler will write the chunk data by taking `dataLength` bytes from each

RTPRssmDecrChunkRefCount

packet, starting at an offset of (<packet data> + transportHeaderLength + payloadHeaderLength).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmDecrChunkRefCount

Tells the base reassembler to dispose of a chunk that it has created or preserved for you.

```
ComponentResult RTPRssmDecrChunkRefCount (
    RTPReassembler    rtpr,
    SHChunkRecord      *inChunk );
```

rtpr

The component instance of the base reassembler component.

inChunk

A pointer to the chunk record to dispose.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

If you have overridden RTPRssmSendPacketList (III–1518) behavior, and are instructing the base reassembler to construct chunks manually, your packet assembler must explicitly dispose of the chunks by calling either this function or RTPRssmSendChunkAndDecrRefCount (III–1517). This function is also used to release a chunk you have preserved using RTPRssmIncrChunkRefCount (III–1513).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmFillPacketListParams

Fills in a packet structure manually.

```
ComponentResult RTPRssmFillPacketListParams (
    RTPReassembler    rtpr,
```

```

RTPRssmPacket    *inPacketListHead,
SInt32           inNumWraparounds,
SInt32           inFlags );

```

rtp

The component instance of the base reassembler.

inPacketListHead

A pointer to the RTPRssmPacket (IV–2412) packet structure.

inNumWraparounds

The high-order 32 bits of the timestamp for this packet. The low-order 32 bits are found in the RTP packet header.

inFlags

A signed 32-bit integer containing any flags (see below) you are passing to the base reassembler.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inFlags Constants

Undocumented

Undocumented

Discussion

Call this function only if your packet reassembler is overriding the RTPRssmSendPacketList (III–1518) behavior. The base reassembler will call back to your packet reassembler using RTPRssmAdjustPacketParams (III–1500) and RTPRssmComputeChunkSize (III–1502).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmGetCapabilities

Obtains the current flag settings for the base reassembler.

```

ComponentResult RTPRssmGetCapabilities (
    RTPReassembler  rtp,
    SInt32           *outFlags );

```

rtp

The component instance of the base reassembler.

RTPRssmGetChunkAndIncrRefCount

outFlags

On return, contains a pointer to the reassembler's current flags (see below).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

outFlags Constants

kRTPRssmEveryPacketAChunkFlag

Tells the base reassembler to treat every packet as a chunk, as would be the case for uLaw audio, for example.

kRTPRssmQueueAndUseMarkerBitFlag

Tells the base reassembler to queue incoming packets and assemble a chunk when a packet has the marker bit set or the RTP time stamp changes.

kRTPRssmTrackLostPacketsFlag

Tells the base reassembler to check the RTP sequence numbers and set the kRTPRssmLostSomePackets flag if any packets are missing from the packet list when it calls your reassembler's RTPRssmSendPacketList (III–1518) function.

kRTPRssmNoReorderingRequiredFlag

Tells the base reassembler that packets do not need to come in sequence-number order, as would be the case for uLaw audio, for example.

Discussion

Your packet reassembler can call this function at any time.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding set function, see RTPRssmSetCapabilities (III–1520).

RTPRssmGetChunkAndIncrRefCount

Causes the base reassembler to create a chunk for you manually.

```
ComponentResult RTPRssmGetChunkAndIncrRefCount (
    RTPReassembler      rtpr,
    UInt32               inChunkDataSize,
    const TimeValue64    *inChunkPresentationTime,
    SHChunkRecord        **outChunk );
```

rtpr

The component instance of the base reassembler component.

`inChunkDataSize`

An unsigned 32-bit integer containing the size of the chunk's data portion, in bytes.

`inChunkPresentationTime`

A pointer to a 64-bit time value specifying the time at which this chunk should be presented, in units of the stream's time scale.

`outChunk`

On return, contains a pointer to a newly-created `SHChunkRecord` (IV–2436) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is useful if you are overriding the `RTPRssmSendPacketList` (III–1518) behavior and constructing the chunk yourself. You must explicitly dispose of the chunk when you are done with it by calling either `RTPRssmDecrChunkRefCount` (III–1504) or `RTPRssmSendChunkAndDecrRefCount` (III–1517).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPRssmGetInfo

Obtains information about your packet reassembler.

```
ComponentResult RTPRssmGetInfo (
    RTPReassembler    rtpr,
    OSType             inSelector,
    void               *ioParams );
```

`rtpr`

The component instance of your packet reassembler.

`inSelector`

A selector (see below) for the information desired.

`ioParams`

A pointer to a data structure appropriate for the type of data requested (see below). If your component understands the selector, write the requested information into the data structure this parameter points to.

RTPRssmGetInfo

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inSelector Constants

kQTSSourceTrackIDInfo

The source track ID as a UInt32; constant value is 'otid'.

kQTSSourceLayerInfo

The source layer number as a UInt16; constant value is 'olyl'.

kQTSSourceLanguageInfo

The source language code as a UInt16; constant value is 'olng'. See “Localization Codes” (IV–2673).

kQTSSourceTrackFlagsInfo

The source track flag set as an SInt32; constant value is 'otfl'.

kQTSSourceDimensionsInfo

The source dimensions as a QTSDimensionParams (IV–2368) structure; constant value is 'odim'.

kQTSSourceVolumesInfo

The source sound volumes as a QTSVolumesParams (IV–2390) structure; constant value is 'ovol'.

kQTSSourceMatrixInfo

The source matrix as a MatrixRecord (IV–2304) structure; constant value is 'omat'.

kQTSSourceClipRectInfo

The source clipping rectangle as a Rect (IV–2399) structure; constant value is 'oclp'.

kQTSSourceGraphicsModeInfo

The source graphics mode as a QTSGraphicsModeParams (IV–2372) structure; constant value is 'ogrm'.

kQTSSourceScaleInfo

The source scaling factors as a Point (IV–2339) structure; constant value is 'oscl'.

kQTSSourceBoundingRectInfo

The source bounding rectangle as a Rect (IV–2399) structure; constant value is 'orct'.

kQTSSourceUserDataInfo

The source’s user data as a UserDataRecord (IV–2496) structure; constant value is 'oudt'.

kQTSSourceInputMapInfo

The source input map as a QT atom container; constant value is 'oimp'.

Discussion

Implement this function only for the selectors you understand. Delegate this function to the base reassembler for any other selectors. The base reassembler will correctly return an error if it doesn't understand the selector either.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding set function, see RTPRssmSetInfo (III–1521). Also see RTPRssmHasCharacteristic (III–1512).

RTPRssmGetPayloadHeaderLength

Obtains the current value of the fixed payload header length from the base reassembler.

```
ComponentResult RTPRssmGetPayloadHeaderLength (
    RTPReassembler    rtp,
    UInt32             *outPayloadHeaderLength );
```

rtp

The component instance of the base reassembler component.

outPayloadHeaderLength

On return, contains a pointer to an unsigned 32-bit integer containing the length of the payload header in bytes. If your packet reassembler does not implement RTPRssmAdjustPacketParams (III–1500), or takes no action, the default *payloadHeaderLength* is the fixed header length that is set (default is 0).

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Discussion

Your packet reassembler can call this function at any time.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding set function, see RTPRssmSetPayloadHeaderLength (III–1523).

RTPRssmGetStreamHandler

RTPRssmGetStreamHandler

Obtains the component instance of the stream handler to which the base reassembler is sending your output.

```
ComponentResult RTPRssmGetStreamHandler (  
    RTPReassembler    rtpr,  
    ComponentInstance *outStreamHandler );
```

rtpr

The component instance of the base reassembler.

outStreamHandler

On return, contains a pointer to the component instance of the stream handler your output is being sent to by the base reassembler.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding set function, see RTPRssmSetStreamHandler (III–1524).

RTPRssmGetTimeScale

Obtains the current time scale from the base reassembler.

```
ComponentResult RTPRssmGetTimeScale (  
    RTPReassembler    rtpr,  
    TimeScale         *outSHTimeScale );
```

rtpr

The component instance of the base reassembler.

outSHTimeScale

On return, contains a pointer to the time scale in use by the stream handler that is processing your output.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

For the corresponding set function, see `RTPRssmSetTimeScale` (III–1525).

RTPRssmGetTimeScaleFromPacket**R**

Lets your packet reassembler extract the time scale from a received packet and return it to the base reassembler.

```
ComponentResult RTPRssmGetTimeScaleFromPacket (
    RTPReassembler    rtpr,
    QTSSStreamBuffer  *inStreamBuffer,
    TimeScale         *outTimeScale );
```

rtpr

The component instance of your packet reassembler.

inStreamBuffer

A pointer to a received packet from which you may be able to extract a time scale.

outTimeScale

Return a pointer to a valid time scale or return an error. If you return a time scale, the packet will be processed normally. If you return an error, the packet will be discarded.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If your packet reassembler has not specified a time scale as part of `RTPRssmNewStreamHandler` (III–1514), or by calling `RTPRssmSetTimeScale` (III–1525), the base reassembler calls this function when it receives packets, which allows your packet reassembler to extract the time scale from a received packet and return it to the base reassembler. Your packet reassembler must set a time scale for the stream handler before the base reassembler can process any incoming packets. If your packet reassembler doesn’t know the time scale of its media in advance, because the time scale is contained in the packet header for example, the base reassembler will prompt you for a time scale whenever it receives a packet. If your packet reassembler always uses the same time scale, it should set the time scale when it opens a stream handler, and it does not need to implement this function. The base reassembler will discard received packets until it has been given a valid time scale.

RTPRssmHandleNewPacket

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmHandleNewPacket

Called whenever a new packet arrives, giving your packet reassembler the opportunity to process the packet.

```
ComponentResult RTPRssmHandleNewPacket (
    RTPReassembler      rtp,
    QTSSStreamBuffer     *inStreamBuffer,
    SInt32               inNumWraparounds );
```

rtp

The component instance of your packet reassembler.

inStreamBuffer

A pointer to the newly-arrived packet.

inNumWraparounds

The upper 32 bits of the 64-bit timestamp (the lower 32 bits are in the RTP packet timestamp).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You should implement this function only if you need to process the packet yourself, or if you need to extract information from the packets as they arrive (you need to monitor the payload header, for example). If you implement this function, you can process the packet as needed, then delegate the default processing to the base reassembler.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmHasCharacteristic

Determines what features your reassembler supports.

```
ComponentResult RTPRssmHasCharacteristic (
    RTPReassembler    rtp,
    OSType             inCharacteristic,
    Boolean            *outHasIt );
```

rtp

The component instance of your packet reassembler.

inCharacteristic

A constant (see below) that defines the characteristic being tested.

outHasIt

A pointer to a Boolean value that is TRUE if your packet reassembler has the characteristic, FALSE otherwise.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inCharacteristic Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

See RTPRssmGetInfo (III–1507) and RTPRssmSetInfo (III–1521).

RTPRssmIncrChunkRefCount

Tells the base reassembler to keep a copy of the most recent chunk after it has been sent.

```
ComponentResult RTPRssmIncrChunkRefCount (
    RTPReassembler    rtp,
    SHChunkRecord     *inChunk );
```

rtp

The component instance of the base reassembler.

inChunk

A pointer to the chunk record you want to preserve.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

RTPRssmInitialize

Discussion

This function is used to assist in loss recovery, for example. You must call `RTPRssmDecrChunkRefCount` (III–1504) to release the chunk when you no longer need it.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPRssmInitialize

Called when the base reassembler is ready to have your packet reassembler begin handling media packets.

```
ComponentResult RTPRssmInitialize (
    RTPReassembler      rtpr,
    RTPRssmInitParams    *inInitParams );
```

rtpr

The component instance of your packet reassembler

inInitParams

A pointer to an `RTPRssmInitParams` (IV–2411) structure. Use the information contained in this structure to initialize your component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is not called when the base reassembler opens your component for payload registration information.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPRssmNewStreamHandler

Opens a new stream handler and closes any currently-open stream handler.

```
ComponentResult RTPRssmNewStreamHandler (
    RTPReassembler      rtpr,
```



```

OSType          inSHType,
SampleDescriptionHandle inSampleDescription,
TimeScale       inSHTimeScale,
ComponentInstance *outHandler );

```

rtptr

The component instance of the base reassembler.

inSHType

The stream handler type.

inSampleDescription

A handle to a `SampleDescription` (IV–2414) structure appropriate for this media type. Pass in `NIL` if you don’t know the media type yet. This structure is passed by reference; the caller is responsible for maintaining it.

inSHTimeScale

The time scale for the stream handler to use. Pass in 0 if the time scale is not yet known.

outHandler

On return, contains a pointer to the component instance of the stream handler that has been opened.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You must pass in a valid `SampleDescription` (IV–2414) structure and time scale before the stream handler can process packets. If you do not pass them as part of this function, do so using `RTPRssmSetTimeScale` (III–1525) and `RTPRssmSetSampleDescription` (III–1524).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPRssmReleasePacketList

Releases memory associated with a packet list that your packet reassembler created itself, or a list your reassembler took ownership of as a result of implementing `RTPRssmSendPacketList` (III–1518).

```

ComponentResult RTPRssmReleasePacketList (
    RTPReassembler  rtptr,

```

RTPRssmReset

```
RTPRssmPacket    *inPacketListHead );
```

rtpr

The component instance of the base reassembler.

inPacketListHead

A pointer to the packet list to dispose of.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This is a housekeeping function that you do not need to perform for packet lists created and handled by the base reassembler, only for packet lists that you create or take ownership of yourself.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPRssmReset

Called to reset all packet reassembler and base reassembler variables for a new run of data.

```
ComponentResult RTPRssmReset (
    RTPReassembler    rtpr,
    SInt32              inFlags );
```

rtpr

The component instance of your reassembler.

inFlags

A signed 32-bit integer containing any flags being passed. No flags are currently defined.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function differs from `RTPRssmClearCachedPackets` (III–1501), which disposes of the packets but still retains the last sequence number and related information; this function resets all variables as if the reassembler were just opened.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

See RTPRssmClearCachedPackets (III–1501).

RTPRssmSendChunkAndDecrRefCount

Called by the packet reassembler when it has finished constructing a chunk and wants the base reassembler to send it to the stream handler.

```
ComponentResult RTPRssmSendChunkAndDecrRefCount (
    RTPReassembler      rtpr,
    SHChunkRecord        *inChunk,
    const SHServerEditParameters *inServerEdit );
```

rtpr

The component instance of the base reassembler.

inChunk

A pointer to an SHChunkRecord (IV–2436) structure.

inServerEdit

A pointer to an SHServerEditParameters (IV–2437) structure containing the server edit parameters. Pass in NIL if there is no server edit.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Use this function to manually send a chunk if you have overridden the default behavior of RTPRssmSendPacketList (III–1518). This function will decrement the reference count of the chunk.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmSendLostChunk

Allows the base reassembler to send loss notification to the stream handler.

```
ComponentResult RTPRssmSendLostChunk (
    RTPReassembler      rtpr,
```

RTPRssmSendPacketList

```
const TimeValue64    *inChunkPresentationTime );
```

rtpr

The component instance of the base reassembler.

inChunkPresentationTime

A pointer to a 64-bit time value indicating when the chunk would have been presented, in units of the stream's time scale.

function result See "Error Codes" (IV–2663). Returns `noErr` if there is no error.

Discussion

Loss notification is normally performed automatically by the base reassembler. Use this function if you are handling losses or sending chunks manually.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPRssmSendPacketList

Called when the base reassembler is ready to send a sample or chunk based on a list of packets.

```
ComponentResult RTPRssmSendPacketList (
    RTPReassembler      rtpr,
    RTPRssmPacket        *inPacketListHead,
    const TimeValue64    *inLastChunkPresentationTime,
    SInt32               inFlags );
```

rtpr

The component instance of your packet reassembler.

inPacketListHead

A pointer to the packet list.

inLastChunkPresentationTime

A pointer to a time value which specifies when to present this chunk, in units of the stream's time scale.

inFlags

A signed 32-bit integer containing any flags being passed (see below).

function result See "Error Codes" (IV–2663). Returns `noErr` if there is no error.

inFlags Constant

kRTPRssmLostSomePackets

A packet loss has occurred.

Discussion

Implement this call if your packet reassembler needs to modify the packet list, or if it overrides the default handling of packet loss. If you do not implement this call, the base reassembler will adjust the packet parameters on all packets in the list, compute the chunk size, and send the chunk. If packet loss has occurred, all the packets will be discarded and the stream handler will be informed that the chunk has been lost.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmSendStreamBufferRange

Notifies the base reassembler to construct and send a chunk based on a part of the stream buffer.

```
ComponentResult RTPRssmSendStreamBufferRange (
    RTPReassembler          rtp,
    RTPSendStreamBufferRangeParams *inParams );
```

rtp

The component instance of the base reassembler.

inParams

A pointer to an RTPSendStreamBufferRangeParams (IV–2413) structure, which specifies the stream buffer, presentation time, start position in the buffer, length of the data in bytes, and any flags.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The contents of the stream buffer will be referenced, not copied. You are responsible for maintaining valid data in the stream buffer.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmSendStreamHandlerChanged

RTPRssmSendStreamHandlerChanged

Called when you have changed something in the stream handler and you want the notification propagated.

```
ComponentResult RTPRssmSendStreamHandlerChanged (  
    RTPReassembler    rtpr );
```

rtpr

The component instance of the base reassembler.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is useful, for example, if you have changed the dimensions of the video.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPRssmSetCapabilities

Sets the capabilities of a streaming packet reassembler.

```
ComponentResult RTPRssmSetCapabilities (  
    RTPReassembler    rtpr,  
    SInt32             inFlags,  
    SInt32             inFlagsMask );
```

rtpr

The component instance of the base reassembler.

inFlags

A signed 32-bit integer containing the logical OR of all the flags (see below) you are setting.

inFlagsMask

Use this field to preserve the state of any flags you do not wish to alter. If a flag (see below) is set in this field, and is not set in the `inFlags` field, it will not be changed from its current setting.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inFlags Constants

kRTPRssmEveryPacketAChunkFlag

Tells the base reassembler to treat every packet as a chunk, as would be the case for uLaw audio, for example.

kRTPRssmQueueAndUseMarkerBitFlag

Tells the base reassembler to queue incoming packets and assemble a chunk when a packet has the marker bit set or the RTP time stamp changes.

kRTPRssmTrackLostPacketsFlag

Tells the base reassembler to check the RTP sequence numbers and set the kRTPRssmLostSomePackets flag if any packets are missing from the packet list when it calls your reassembler's RTPRssmSendPacketList (III–1518) function.

kRTPRssmNoReorderingRequiredFlag

Tells the base reassembler that packets do not need to come in sequence-number order, as would be the case for uLaw audio, for example.

Discussion

Your packet reassembler can call this function at any time.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding get function, see RTPRssmGetCapabilities (III–1505).

RTPRssmSetInfo

Sets various parameters of your packet reassembler; it is also called to set parameters of the base reassembler.

```
ComponentResult RTPRssmSetInfo (
    RTPReassembler    rtpr,
    OSType             inSelector,
    void               *ioParams );
```

rtpr

The component instance of your packet reassembler.

inSelector

A selector (see below) for the information being set. Ignore any selectors you do not understand.

RTPRssmSetInfo

ioParams

A pointer to the information that should be set.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

inSelector Constants

`kQTSSourceTrackIDInfo`

The source track ID as a `UInt32`; constant value is `'otid'`.

`kQTSSourceLayerInfo`

The source layer number as a `UInt16`; constant value is `'olyr'`.

`kQTSSourceLanguageInfo`

The source language code as a `UInt16`; constant value is `'olng'`. See “Localization Codes” (IV–2673).

`kQTSSourceTrackFlagsInfo`

The source track flag set as an `SInt32`; constant value is `'otfl'`.

`kQTSSourceDimensionsInfo`

The source dimensions as a `QTSDimensionParams` (IV–2368) structure; constant value is `'odim'`.

`kQTSSourceVolumesInfo`

The source sound volumes as a `QTSVolumesParams` (IV–2390) structure; constant value is `'ovol'`.

`kQTSSourceMatrixInfo`

The source matrix as a `MatrixRecord` (IV–2304) structure; constant value is `'omat'`.

`kQTSSourceClipRectInfo`

The source clipping rectangle as a `Rect` (IV–2399) structure; constant value is `'oclp'`.

`kQTSSourceGraphicsModeInfo`

The source graphics mode as a `QTSGraphicsModeParams` (IV–2372) structure; constant value is `'ogrm'`.

`kQTSSourceScaleInfo`

The source scaling factors as a `Point` (IV–2339) structure; constant value is `'oscl'`.

`kQTSSourceBoundingRectInfo`

The source bounding rectangle as a `Rect` (IV–2399) structure; constant value is `'orct'`.

kQTSSourceUserDataInfo

The source's user data as a UserDataRecord (IV–2496) structure; constant value is 'oudt'.

kQTSSourceInputMapInfo

The source input map as a QT atom container; constant value is 'oimp'.

Discussion

Delegate this function to the base reassembler for any selectors you don't understand. If the base reassembler doesn't understand them either, it will return an error to the caller.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding get function, see RTPRssmGetInfo (III–1507). Also see RTPRssmHasCharacteristic (III–1512).

RTPRssmSetPayloadHeaderLength

Called by the packet reassembler to set a fixed header length for your payload.

```
ComponentResult RTPRssmSetPayloadHeaderLength (
    RTPReassembler    rtpr,
    UInt32             inPayloadHeaderLength );
```

rtpr

The component instance of the base reassembler.

inPayloadHeaderLength

An unsigned 32-bit integer containing the fixed payload header length, in bytes. If your packet reassembler does not implement RTPRssmAdjustPacketParams (III–1500), or takes no action, the default payloadHeaderLength is the fixed header length that is set (default is 0).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmSetSampleDescription

See Also

For the corresponding get function, see RTPRssmGetPayloadHeaderLength (III–1509).

RTPRssmSetSampleDescription

Changes the SampleDescription (IV–2414) structure being used by the stream handler; all subsequent samples will be marked with this new structure.

```
ComponentResult RTPRssmSetSampleDescription (  
    RTPReassembler      rtpR,  
    SampleDescriptionHandle inSampleDescription );
```

rtpR

The component instance of the base reassembler.

inSampleDescription

The handle of a SampleDescription (IV–2414) structure to use. You are responsible for keeping the handle and the data structure valid during subsequent operations.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The SampleDescription structure is not passed on a per-packet basis, but a per-sample basis, so the SampleDescription (IV–2414) structure should not be changed until a complete sample (sometimes called a “frame” or “chunk”) has been reassembled.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmSetStreamHandler

Assigns a stream handler to the output of the base reassembler.

```
ComponentResult RTPRssmSetStreamHandler (  
    RTPReassembler      rtpR,  
    ComponentInstance    inStreamHandler );
```

rtpR

The component instance of the base reassembler.

`inStreamHandler`

The component instance of the stream handler to use.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The stream handler must already be opened and initialized, and its time scale must already be set.

Special Considerations

Use this function only if you have opened and initialized a stream handler yourself. The base reassembler will not close the stream handler it is already using.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QTStreamingComponents.h`

See Also

For the corresponding get function, see `RTPRssmGetStreamHandler` (III–1510).

RTPRssmSetTimeScale

Sets the time scale for the stream handler that will render your output.

```
ComponentResult RTPRssmSetTimeScale (
    RTPReassembler    rtpr,
    TimeScale          inSHTimeScale );
```

`rtpr`

The component instance of the base reassembler

`inSHTimeScale`

The time scale for the stream handler to use

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The time scale is the number of time units that pass in one second for the media whose sample data is carried in this stream. The stream handler’s time scale must be set before it can deliver any data to the user.

Special Considerations

This function is normally used by a packet reassembler when the time scale to use is not initially known. You don’t need to call this function if you specified a time scale when the stream handler was opened.

SampleNumToMediaTime

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QTStreamingComponents.h

See Also

For the corresponding get function, see RTPRssmGetTimeScale (III–1510).

S

SampleNumToMediaTime

Finds the time at which a specified sample plays.

```
void SampleNumToMediaTime (
    Media          theMedia,
    long           logicalSampleNum,
    TimeValue      *sampleTime,
    TimeValue      *sampleDuration );
```

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

logicalSampleNum

The sample number.

sampleTime

A pointer to a time value. The function updates this time value to indicate the starting time of the sample specified by the `logicalSampleNum` parameter. This time value is expressed in the media's time scale. Set this parameter to `NIL` if you don't want this information.

sampleDuration

A pointer to a time value. The Movie Toolbox returns the duration of the sample specified by the `logicalSampleNum` parameter. This time value is expressed in the media's time scale. Set this parameter to `NIL` if you don't want this information.

function result You can access this function's error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Samples” (V–2742)

Related Java Methods

`quicktime.std.movies.media.Media.sampleNumToMediaTime()`

ScaleMatrix

Modifies the contents of a matrix so that it defines a scaling operation.

```
void ScaleMatrix (
    MatrixRecord    *m,
    Fixed           scaleX,
    Fixed           scaleY,
    Fixed           aboutX,
    Fixed           aboutY );
```

`m`

A pointer to a `MatrixRecord` (IV–2304) structure. The `ScaleMatrix` function updates the contents of this matrix so that the matrix describes a scaling operation; that is, it concatenates the respective transformations onto whatever was initially in the matrix structure. You specify the magnitude of the scaling operation with the `scaleX` and `scaleY` parameters. You specify the anchor point with the `aboutX` and `aboutY` parameters.

`scaleX`

The scaling factor applied to x coordinates.

`scaleY`

The scaling factor applied to y coordinates.

`aboutX`

The x coordinate of the anchor point.

`aboutY`

The y coordinate of the anchor point.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Matrices” (V–2764)

SCAsyncIdle

Related Java Methods

`quicktime.std.image.Matrix.scale()`

SCAsyncIdle

Called occasionally while performing asynchronous compression with `SCCompressSequenceFrameAsync` (III–1538).

`ComponentResult SCAsyncIdle (ComponentInstance ci);`

`ci`

Your application’s connection to the image-compression component being used by `SCCompressSequenceFrameAsync` (III–1538). You obtain this identifier from `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `SCCompressSequenceFrameAsync` (III–1538).

ScaleMovieSegment

Changes the duration of a segment of a movie.

```
OSErr ScaleMovieSegment (
    Movie          theMovie,
    TimeValue      startTime,
    TimeValue      oldDuration,
    TimeValue      newDuration );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), or `NewMovieFromHandle` (II–1084).

`startTime`

The start of the segment. The `oldDuration` parameter specifies the segment’s duration. This time value must be expressed in the movie’s time scale.

`oldDuration`

The original duration of the segment in the source movie. This time value must be expressed in the movie's time scale.

`newDuration`

The new duration of the segment. This time value must be expressed in the movie's time scale. The function alters the segment to accommodate the new duration.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

The Movie Toolbox scales the segment to accommodate the new duration.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Low-Level Movie Editing Functions” (V-2749)

Related Java Methods

`quicktime.std.movies.Movie.scaleSegment()`

ScaleTrackSegment

Changes the duration of a segment of a track.

```
OSErr ScaleTrackSegment (
    Track          theTrack,
    TimeValue      startTime,
    TimeValue      oldDuration,
    TimeValue      newDuration );
```

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II-1092) and `GetMovieTrack` (I-497).

`startTime`

The start of the segment. The `oldDuration` parameter specifies the segment's duration. This time value must be expressed in the time scale of the movie that contains the track.

SCCompressImage

`oldDuration`

The duration of the segment. This time value must be expressed in the time scale of the movie that contains the track.

`newDuration`

The new duration of the segment. This time value must be expressed in the time scale of the movie that contains the track. The function alters the segment to accommodate the new duration.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

This function does not cause the Movie Toolbox to add data to or remove data from the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Editing Tracks” (V–2750)

Related Java Methods

`quicktime.std.movies.Track.scaleSegment()`

SCCompressImage

Compresses an image that is stored in a `PixelFormat` (IV–2332) structure.

```
ComponentResult SCCompressImage (
    ComponentInstance      ci,
    PixelMapHandle         src,
    const Rect              *srcRect,
    ImageDescriptionHandle *desc,
    Handle                  *data );
```

`ci`

Identifies your application’s connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

`src`

A handle to the `PixelFormat` (IV–2332) structure to be compressed.

srcRect

A pointer to a portion of the `PixelFormat` (IV–2332) structure to compress as a `Rect` (IV–2399) structure. This rectangle must be in the pixel map’s coordinate system. If you want to compress the entire pixel map, set this parameter to `NIL`.

desc

A pointer to a handle to an `ImageDescription` (IV–2285) structure. The standard dialog component creates an `ImageDescription` structure when it compresses the image, and returns a handle to that structure in the field referred to by this parameter. The component sizes that handle appropriately. Your application is responsible for disposing of that handle when you are done with it.

data

A pointer to a handle. The standard dialog component returns a handle to the compressed image data in the field referred to by this parameter. The component sizes that handle appropriately. Your application is responsible for disposing of that handle when you are done with it.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Compressing Still Images” (V–2801)

Related Java Methods

```
quicktime.std.qtcomponents.SCSequence.compressFrame(),
quicktime.std.qtcomponents.ImageCompressionDialog.compressImage(),
quicktime.std.image.ImageDescription.fromImageCompressionDialog()
```

SCCompressPicture

Compresses a `Picture` (IV–2331) structure that is stored by a handle.

```
ComponentResult SCCompressPicture (
    ComponentInstance ci,
    PicHandle        srcPicture,
    PicHandle        dstPicture );
```

SCCompressPictureFile

`ci`

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II-1131).

`srcPicture`

A handle to the `Picture` (IV-2331) structure to be compressed.

`dstPicture`

A handle to the compressed `Picture` (IV-2331) structure. The standard dialog component resizes this handle to accommodate the compressed structure. Your application is responsible for creating and disposing of this handle when you are done with it.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Compressing Still Images” (V-2801)

Related Java Methods

`quicktime.qd.Pict.fromImageCompressionDialog()`,

`quicktime.std.qtcomponents.ImageCompressionDialog.compressPicture()`

SCCompressPictureFile

Compresses a `Picture` (IV-2331) structure that is stored in a file.

```
ComponentResult SCCompressPictureFile (  
    ComponentInstance    ci,  
    short                srcRefNum,  
    short                dstRefNum );
```

`ci`

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II-1131).

`srcRefNum`

A reference to the file to be compressed.

`dstRefNum`

A reference to the file that is to receive the compressed data. This may be the same as the source file. The standard dialog component places the compressed image data into the file identified by this reference. Your application is responsible for this file after the compression operation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Compressing Still Images” (V–2801)

Related Java Methods

`quicktime.std.qtcomponents.ImageCompressionDialog.compressPictureFile()`

SCCompressSequenceBegin

Initiates a sequence-compression operation.

```
ComponentResult SCCompressSequenceBegin (
    ComponentInstance      ci,
    PixMapHandle           src,
    const Rect              *srcRect,
    ImageDescriptionHandle *desc );
```

`ci`

Identifies your application’s connection to a standard image-compression component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

`src`

A handle to the `PixMap` (IV–2332) structure to be compressed. This pixel map must contain the first image in the sequence.

`srcRect`

A pointer to a portion of the `PixMap` (IV–2332) structure to compress as a `Rect` (IV–2399) structure. This rectangle must be in the pixel map’s coordinate system. If you want to compress the entire structure, set this parameter to `NIL`.

`desc`

A pointer to an image description handle. The standard dialog component creates an image description structure when it compresses the image, and returns a handle to that structure in the field referred to by this parameter. The

SCCompressSequenceEnd

component sizes the handle appropriately. If you do not want this information, set this parameter to NIL.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Compressing Image Sequences” (V–2802)

Related Java Methods

quicktime.std.qtcomponents.ImageCompressionDialog.compressSequenceBegin()

SCCompressSequenceEnd

Ends a sequence-compression operation.

```
ComponentResult SCCompressSequenceEnd (  
    ComponentInstance    ci );
```

ci

Identifies your application’s connection to a standard image-compression component. You obtain this identifier from OpenDefaultComponent (II–1131).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The standard dialog component disposes of any memory it used to compress the image sequence, including the data and image description buffers. You must call this function once for each sequence you start.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Compressing Image Sequences” (V–2802)

SCCompressSequenceFrame

Continues a sequence-compression operation.

```
ComponentResult SCCompressSequenceFrame (
```

```
ComponentInstance    ci,
PixMapHandle         src,
const Rect           *srcRect,
Handle               *data,
long                  *dataSize,
short                 *notSyncFlag );
```

ci

Identifies your application's connection to a standard image-compression component. You obtain this identifier from `OpenDefaultComponent` (II-1131).

src

A handle to the `PixMap` (IV-2332) structure to be compressed.

srcRect

A pointer to a portion of the `PixMap` (IV-2332) structure to compress as a `Rect` (IV-2399) structure. This rectangle must be in the pixel map's coordinate system. If you want to compress the entire pixel map, set this parameter to `NIL`.

data

A pointer to a handle. The standard dialog component returns a handle to the compressed image data in the field referred to by this parameter. The component sizes that handle appropriately for the sequence.

dataSize

A pointer to a long integer. The standard dialog component returns a value that indicates the number of bytes of compressed image data that it returns. Note that this value will differ from the size of the handle referred to by the `data` parameter, because the handle is allocated to accommodate the largest image in the sequence.

notSyncFlag

A pointer to a short integer that indicates whether the compressed frame is a **key frame**. If the frame is a key frame, the standard dialog component sets the field referred to by this parameter to 0; otherwise, the component sets this field to `mediaSampleNotSync`. You may use this field to set the `sampleFlags` parameter of the `AddMediaSample` (I-27) function.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

You must call this function once for each frame in the sequence, including the first frame. The following is an example of its use:

```
// SCCompressSequenceFrame coding example
// See "Discovering QuickTime," page 246
#define kSampleDuration                240
```

SCCompressSequenceFrame

```
                                // video frames last 240 * 1/600th of a second
#define kNumVideoFrames          29
#define kNoOffset                0
#define kMgrChoose               0
#define kSyncSample              0
#define kAddOneVideoSample       1
#define kPixelDepth              16

void AddMyVideoSamplesToMedia (Media media, const Rect *rect)
{
    long                lMaxCompressedSize;
    GWorldPtr           pGWorld = NIL;
    long                lCurSample;
    Handle              hCompressedData = NIL;
    Ptr                 pCompressedData;
    ImageDescriptionHandle hImageDesc = NIL;
    CGrafPtr            pOldPort;
    GDHandle             hghOldDevice;
    OSErr               nErr = noErr;
    ComponentInstance    stdComponent = NIL;
    ComponentResult      lErr = noErr;

    nErr = NewGWorld(&pGWorld,
                    kPixelDepth,
                    rect,
                    NIL,
                    NIL,
                    (GWorldFlags)0);

    LockPixels(pGWorld->portPixMap);

    nErr = GetMaxCompressionSize(pGWorld->portPixMap,
                                rect,
                                0,          // let ICM choose depth
                                codecNormalQuality,
                                kAnimationCodecType,
                                (CompressorComponent)anyCodec,
                                &lMaxCompressedSize);

    hCompressedData = NewHandle(lMaxCompressedSize);
```

```

MoveHHI(hCompressedData);
HLock(hCompressedData);
pCompressedData = StripAddress(*hCompressedData);

hImageDesc = (ImageDescriptionHandle)NewHandle(4);
GetGWorld(&pOldPort, &hghOldDevice);
SetGWorld(pGWorld, NIL);

nErr = OpenADefaultComponent(StandardCompressionType,
                             StandardCompressionSubType,
                             stdComponent);

lErr = SCCompressSequenceBegin(stdComponent,
                               pGWorld->portPixMap,
                               rect,
                               &hImageDesc);

for (lCurSample = 1; lCurSample < 30; lCurSample++) {
    short    syncFlag;
    EraseRect(rect);
    MyDrawAFrame(rect, lCurSample);

    lErr = SCCompressSequenceFrame(stdComponent,
                                   pGWorld->portPixMap,
                                   rect,
                                   &pCompressedData,
                                   &(**hImageDesc).dataSize,
                                   &syncFlag);

    nErr = AddMediaSample(media,
                          hCompressedData,
                          0,           // no offset in data
                          (**hImageDesc).dataSize,
                          60,         // frame duration = 1/10 sec
                          (SampleDescriptionHandle)hImageDesc,
                          1,          // one sample
                          syncFlag,   // key frame or not?
                          NIL);
}
SetGWorld(pOldPort, hghOldDevice);
lErr = SCCompressSequenceEnd(stdComponent);

```

SCCompressSequenceFrameAsync

```
    if (lErr != NIL)
        CloseComponent(stdComponent);
    if (hImageDesc != NIL)
        DisposeHandle((Handle)hImageDesc);
    if (hCompressedData != NIL)
        DisposeHandle(hCompressedData);
    if (pGWorld != NIL)
        DisposeGWorld(pGWorld);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Compressing Image Sequences” (V–2802)

Related Java Methods

`quicktime.util.QTHandleRef.fromSCSequence()`

See Also

For an asynchronous version of this function, with a completion callback, see `SCCompressSequenceFrameAsync` (III–1538).

SCCompressSequenceFrameAsync

An asynchronous variant of `SCCompressSequenceFrame` (III–1534), with a completion callback.

```
ComponentResult SCCompressSequenceFrameAsync (
    ComponentInstance      ci,
    PixMapHandle           src,
    const Rect             *srcRect,
    Handle                 *data,
    long                   *dataSize,
    short                  *notSyncFlag,
    ICMCompletionProcRecordPtr  asyncCompletionProc);
```

`ci`

Identifies your application’s connection to a standard image-compression component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

`src`

A handle to the `PixMap` (IV–2332) structure to be compressed.

srcRect

A pointer to a portion of the `PixelFormat` (IV–2332) structure to compress as a `Rect` (IV–2399) structure. This rectangle must be in the pixel map’s coordinate system. If you want to compress the entire pixel map, set this parameter to `NIL`.

data

A pointer to a handle. The standard dialog component returns a handle to the compressed image data in the field referred to by this parameter. The component sizes that handle appropriately for the sequence.

dataSize

A pointer to a long integer. The standard dialog component returns a value that indicates the number of bytes of compressed image data that it returns. Note that this value will differ from the size of the handle referred to by the `data` parameter, because the handle is allocated to accommodate the largest image in the sequence.

notSyncFlag

A pointer to a short integer that indicates whether the compressed frame is a **key frame**. If the frame is a key frame, the standard dialog component sets the field referred to by this parameter to 0; otherwise, the component sets this field to `mediaSampleNotSync`. You may use this field to set the `sampleFlags` parameter of the `AddMediaSample` (I–27) function.

asyncCompletionProc

A pointer to an `ICMCompletionProcRecord` (IV–2279) structure. If you pass `NIL`, the `SCCompressSequenceFrameAsync` function acts like `SCCompressSequenceFrame` (III–1534).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

While performing asynchronous compression with this function, you should occasionally call `SCAsyncIdle` (III–1528). This gives the standard compression component an opportunity to restart its compression operation if it needs to force a **key frame**.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `SCCompressSequenceFrame` (III–1534).

SCDefaultPictFileSettings

SCDefaultPictFileSettings

Derives default compression settings for a `Picture` (IV–2331) structure that is stored in a file.

```
ComponentResult SCDefaultPictFileSettings (  
    ComponentInstance    ci,  
    short                srcRef,  
    short                motion );
```

`ci`

Identifies your application’s connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

`srcRef`

A reference to the file to be analyzed.

`motion`

Specifies whether the image is part of a sequence. Set this parameter to `TRUE` if the image is part of a sequence; set it to `FALSE` if you are working with a single still image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Getting Default Settings for an Image or a Sequence” (V–2800)

Related Java Methods

`quicktime.std.qtcomponents.ImageCompressionDialog.defaultPictFileSettings()`

SCDefaultPictHandleSettings

Derives default compression settings for a `Picture` (IV–2331) structure that is stored by a handle.

```
ComponentResult SCDefaultPictHandleSettings (  
    ComponentInstance    ci,  
    PicHandle            srcPicture,  
    short                motion );
```

`ci`

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II-1131).

`srcPicture`

A handle to the `Picture` (IV-2331) structure to be analyzed.

`motion`

Specifies whether the image is part of a sequence. Set this parameter to `TRUE` if the image is part of a sequence; set it to `FALSE` if you are working with a single still image.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: "Getting Default Settings for an Image or a Sequence" (V-2800)

Related Java Methods

`quicktime.std.qtcomponents.ImageCompressionDialog.defaultPictSettings()`

SCDefaultPixMapSettings

Derives default compression settings for an image that is stored in a pixel map.

```
ComponentResult SCDefaultPixMapSettings (
    ComponentInstance    ci,
    PixMapHandle         src,
    short                motion );
```

`ci`

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II-1131).

`src`

A handle to the `PixMap` (IV-2332) structure to be analyzed.

SCGetBestDeviceRect

motion

Specifies whether the image is part of a sequence. Set this parameter to `TRUE` if the image is part of a sequence; set it to `FALSE` if you are working with a single still image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Getting Default Settings for an Image or a Sequence” (V–2800)

Related Java Methods

`quicktime.std.qtcomponents.ImageCompressionDialog.defaultPixmapSettings()`

SCGetBestDeviceRect

Determines the boundary rectangle that surrounds the display device that supports the largest color or grayscale palette.

```
ComponentResult SCGetBestDeviceRect (  
    ComponentInstance ci,  
    Rect *r );
```

ci

Identifies your application’s connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

r

A pointer to a `Rect` (IV–2399) structure. The function returns the global coordinates of a rectangle that surrounds the appropriate display device.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The standard image-compression dialog component uses this function to position rectangles and dialog boxes when you indicate that the component is to choose the best display device. It subtracts the menu bar from the returned rectangle if the best device is also the main display device.

Special Considerations

In general, your application does not need to use this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Positioning Dialog Boxes and Rectangles” (V–2803)

SCGetCompressFlags

Gets compression flags for a standard image-compression dialog component.

```
ComponentResult SCGetCompressFlags (
    ComponentInstance ci,
    long *flags );
```

ci

Identifies your application’s connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

flags

A pointer to compression flags (see below).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`scCompressFlagIgnoreIdenticalFrames`
Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Related Java Methods

`quicktime.std.qtcomponents.ImageCompressionDialog.getCompressFlags()`

See Also

For the corresponding set function, see `SCSetCompressFlags` (III–1557).

SCGetCompressionExtended

SCGetCompressionExtended

Undocumented

```
ComponentResult SCGetCompressionExtended (
    ComponentInstance ci,
    SCParams          *params,
    Point             where,
    SModalFilterUPP   filterProc,
    SModalHookUPP     hookProc,
    long              refcon,
    StringPtr         customName );
```

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

params

A pointer to an `SCParams` (IV–2420) structure.

where

Undocumented

filterProc

A **Universal Procedure Pointer** that accesses a `SModalFilterProc` (III–2133) callback.

hookProc

A **Universal Procedure Pointer** that accesses a `SModalHookProc` (III–2134) callback.

refcon

A reference constant to be passed to your callbacks. Use this parameter to point to a data structure containing any information your callbacks need.

customName

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

SCGetInfo

Retrieves configuration information from the standard dialog component.

```
ComponentResult SCGetInfo (
    ComponentInstance ci,
    OSType           infoType,
    void             *info );
```

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II-1131).

infoType

A constant (see below) that specifies the type of information you want to retrieve.

info

A pointer to a field that is to receive the information. The `infoType` constant descriptions (see below) include information about this field.

function result See “Error Codes” (IV-2663). If the component cannot satisfy your request, it returns a result code of `scTypeNotFoundErr`. It returns `noErr` if there is no error.

infoType Constants

`scSpatialSettingsType`

The component's spatial compression parameters in an `SCSpatialSettings` (IV-2423) structure.

`scTemporalSettingsType`

The component's temporal compression parameters in an `SCTemporalSettings` (IV-2427) structure.

`scDataRateSettingsType`

Information about the component's compression data rate in an `SCDataRateSettings` (IV-2417) structure.

`scColorTableType`

The component's color table as a pointer to a `CTabHandle`, which references a `ColorTable` (IV-2210) structure.

`scProgressProcType`

A pointer to a `ICMProgressProcRecord` (IV-2284) structure. Set the pointer to `NIL` to clear the current **progress function**; in this case, the standard dialog

component does not report its progress to the user. Set the pointer to -1 to use the component's default progress function.

`scExtendedProcsType`

A pointer to an `SCExtendedProcs` (IV-2418) structure.

`scPreferenceFlagsType`

A pointer to a long integer. It contains constants (see below) that define preference flags.

`scSettingsStateType`

A pointer to a handle to the component's complete configuration. Your application is responsible for disposing of the handle when you are done with it. Set the pointer to NIL to reset the component to its default configuration.

Your application is not concerned with the content of the configuration information that is returned. The standard dialog component saves its configuration in a format that it understands. This request affects only those settings that are valid across system restarts, such as the spatial and temporal compression parameters and the data-rate settings. When you retrieve the settings, the standard dialog component creates an appropriately-sized handle and places its current configuration information into the handle. When you modify the settings, you supply the configuration information in the handle, and the component copies the data out of this handle.

`scSequenceIDType`

The component's current image-compression sequence identifier being used by the component's `SCCompressSequenceFrame` (III-1534) function. You supply a pointer to a field of type `ImageSequence`; see "Codec Types" (IV-2619). The standard dialog component returns the current sequence identifier in that field.

`scWindowPositionType`

A pointer to a `Point` (IV-2339) structure. If you are retrieving this information, the standard dialog component places the coordinates of the upper-left corner of the dialog box into this structure; if you are changing the dialog box's position, place the new coordinates into the structure; the component uses those coordinates to position the dialog box.

Normally you should not need to use this request. By default, the standard dialog component centers the dialog box on the screen that is best-suited to display your test image. The component also saves the last window position for movable modal dialogs.

scCodecFlagsType

A pointer to a field of type `CodecFlags` that contains flag constants (see below); see “Codec Types” (IV–2619). If you are retrieving the flags, the standard dialog component places the current flags into this field. If you are setting new flag values, place your desired settings into the field; the component uses these new flag settings.

By default, the standard dialog component sets all flags to 0 when it compresses still images. When it is compressing sequences, the component sets the `codecFlagUpdatePrevious` and `codecFlagUpdatePreviousComp` flags to 1. Typically, you should not need to change these flag settings.

scPreferenceFlagsType Constants**scListEveryCodec**

Controls the contents of the pop-up menu of compressors. If you set this flag to 1, the standard image-compression dialog component lists every compressor component that is present in the system. Each entry in the list contains the name of a compressor component. The user may then select a specific component from the list. If you set this flag to 0, the list contains one entry for each type of compressor component that is present in the system. Each list entry contains the name of a compressor type (for example, a list entry might contain “Animation” for the animation compressor). The user may then select a type of compressor; it is your application’s responsibility to select an appropriate compressor of that type.

scAllowZeroFrameRate

Determines whether the component allows the user to specify a value of 0 for the frame rate. If you set this flag to 1, the component allows the user to specify either 0 or nothing for the frame rate. The component then includes a “best rate” entry in the pop-up menu. If the user specifies 0, the component sets the `frameRate` field in the `STemporalSettings` (IV–2427) structure to 0. Your application must then determine the best frame rate for the movie. If you set this flag to 0, the component does not allow the user to enter 0 for the frame rate. In this case, the user must select a specific frame rate.

scAllowZeroKeyFrameRate

Similar to the `scAllowZeroFrameRate` flag, this flag determines whether the component allows the user to specify a value of 0 for the **key frame** rate. If you set this flag to 1, the component allows the user to specify 0 for the frame rate. If the user specifies 0, the component sets the `keyFrameRate` field in the `STemporalSettings` (IV–2427) structure to 0. Your application must then determine the best key frame rate for the movie. If you set this flag to 0, the component does not allow the user to specify 0 for the frame rate. In this case,

if the user has enabled temporal compression by checking the key frame checkbox, the user must also select a specific key frame rate.

`scShowBestDepth`

Determines whether the component includes a “best depth” entry in the pop-up menu for pixel depth. If you set this flag to 1, the component includes a “best depth” entry in the pop-up menu. If the user selects “best depth,” the component sets the depth to 0. Your application must then determine the best pixel depth for the movie. If you set this flag to 0, the component does not include a “best depth” entry in the pop-up menu. The user must select a depth from among the other available choices.

`scUseMovableModal`

Determines whether the standard compression dialog is a movable or a stationary dialog. Set this flag to 1 to create a movable dialog. In this case, you should provide an event filter function to handle update events; use the `scExtendedProcsType` request (see above).

scCodecFlagsType Constants

`codecFlagUseImageBuffer`

Controls whether the decompressor allocates an offscreen buffer for decompression. If your application sets this flag to 1, the decompressor allocates an offscreen buffer the size of the compressed image. If you set this flag to 0, the decompressor does not use an offscreen image buffer. These image buffers are useful when decompressing sequences that were created using temporal compression.

`codecFlagUseScreenBuffer`

Controls whether the decompressor allocates an offscreen destination buffer during decompression. If you set this flag to 1, the decompressor allocates an offscreen buffer the size of the destination screen. If you set this flag to 0, the decompressor does not use an offscreen screen buffer. Using a screen buffer helps to reduce tearing that may result when decompressing directly to the screen.

`codecFlagUpdatePrevious`

Controls whether the compressor updates the previous image buffer during compression. This flag is only used with sequences that are being temporally compressed. If you set this flag to 1, the compressor copies the current source image into the previous frame buffer at the end of the frame compression.

`codecFlagNoScreenUpdate`

Controls whether the decompressor updates the screen image. If you set this flag to 1, the decompressor does not write the current frame to the screen, but

does write the frame to its offscreen image buffer (if one was allocated). If you set this flag to 0, the decompressor writes the frame to the screen.

`codecFlagWasCompressed`

Indicates to the compressor that the image to be compressed has been compressed before. This information may be useful to compressors that can compensate for the image degradation that may otherwise result from repeated compression and decompression of the same image. Set this flag to 1 to indicate that the image was previously compressed. Set this flag to 0 if the image was not previously compressed.

`codecFlagDontOffscreen`

Controls whether the decompressor uses the offscreen buffer during sequence decompression. This flag is only used with sequences that have been temporally compressed. If this flag is set to 1, the decompressor does not use the offscreen buffer during decompression. Instead, the decompressor returns an error. This allows your application to refill the offscreen buffer. If this flag is set to 0, the decompressor uses the offscreen buffer if appropriate.

`codecFlagUpdatePreviousComp`

Controls whether the compressor updates the previous image buffer with the decompressed image data. This flag is only used with temporal compression and is similar to the `codecFlagUpdatePrevious` flag. As with the `codecFlagUpdatePrevious` flag, if you set this flag to 1, the compressor updates the previous frame buffer at the end of the frame compression. However, this flag causes the Image Compression Manager to update the frame buffer using an image obtained by decompressing the results of the most recent compression operation, rather than the source image.

`codecFlagForceKeyFrame`

Controls whether the compressor creates a **key frame** from the current image. This flag is only used with temporal compression. If you set this flag to 1, the compressor makes the current image a key frame. If you set this flag to 0, the compressor decides based on other criteria, such as the key frame rate, whether to create a key frame from the current image.

`codecFlagOnlyScreenUpdate`

Controls whether the decompressor decompresses the current frame. If you set this flag to 1, the decompressor writes the contents of its offscreen image buffer to the screen, but does not decompress the current frame. If you set this flag to 0, the decompressor decompresses the current frame and writes it to the screen. You can set this flag to 1 only if you have allocated an offscreen image buffer for use by the decompressor.

SCGetInfo

`codecFlagLiveGrab`

Indicates to the compressor whether the current sequence results from grabbing live video. When working with live video, compressors operate as quickly as possible and disable some additional processing, such as compensation for previously compressed data. Set this flag to 1 when you are compressing from a live video source; the compressor then operates as quickly as it can.

`codecFlagUsedNewImageBuffer`

Indicates to your application that the decompressor used the offscreen image buffer for the first time when it processed this frame. If this flag is set to 1, the decompressor used the image buffer for this frame and this is the first time the decompressor used the image buffer in this sequence. If this flag is set to 0, the decompressor did not use the image buffer.

`codecFlagUsedImageBuffer`

Indicates to your application that the decompressor used the offscreen image buffer for this frame. If this flag is set to 1, the decompressor used the image buffer. If this flag is set to 0, the decompressor did not use the image buffer.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing the Sound Dialog” (V–2895), “Working With Image or Sequence Settings” (V–2802)

Related Java Methods

```
quicktime.qd.ColorTable.fromCompressionDialog()  
quicktime.std.qtcomponents.ImageCompressionDialog.getInfoSpatialSettings()  
quicktime.std.qtcomponents.ImageCompressionDialog.getInfoTemporalSettings()  
quicktime.std.qtcomponents.ImageCompressionDialog.getInfoDataRateSettings()  
quicktime.std.qtcomponents.ImageCompressionDialog.getInfoColorTable()  
quicktime.std.qtcomponents.CompressionDialog.getInfoState()  
quicktime.std.qtcomponents.CompressionDialog.getInfoPreferences()  
quicktime.std.qtcomponents.SoundCompressionDialog.getInfoSampleRate()  
quicktime.std.qtcomponents.SoundCompressionDialog.getInfoSampleSize()  
quicktime.std.qtcomponents.SoundCompressionDialog.getInfoChannelCount()  
quicktime.std.qtcomponents.SoundCompressionDialog.getInfoCompression()  
quicktime.std.qtcomponents.SoundCompressionDialog.getInfoCompressionList()
```

See Also

For the corresponding set function, see `SCSetInfo` (III–1558).

SCGetSettingsAsAtomContainer

Places the current configuration from the standard image-compression component in a QT atom container.

```
ComponentResult SCGetSettingsAsAtomContainer (
    ComponentInstance ci,
    QTAtomContainer *settings );
```

ci

The standard compression component instance.

settings

The address where the newly-created atom container should be stored.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The caller is responsible for disposing of the returned QT atom container.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Related Java Methods

`quicktime.std.movies.AtomContainer.fromCompressionDialog()`

SCGetSettingsAsText

Undocumented

```
ComponentResult SCGetSettingsAsText (
    ComponentInstance ci,
    Handle *text );
```

ci

Identifies your application’s connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

text

A pointer to a handle to text.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

SCNewGWorld

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

SCNewGWorld

Creates a graphics world based on the current compression settings.

```
ComponentResult SCNewGWorld (
    ComponentInstance ci,
    GWorldPtr *gwp,
    Rect *rp,
    GWorldFlags flags );
```

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II-1131).

gwp

A pointer to a pointer to a `CGrafPort` (IV-2168) structure that defines a graphics world. The standard dialog component places a pointer to the new graphics world into the field referred to by this parameter. If the component cannot create the graphics world, it sets this field to `NIL`.

rp

A pointer to the boundaries of the graphics world. If you set this parameter to `NIL`, the standard dialog component uses the test image's boundary rectangle. If you don't specify a boundary rectangle and there is no test image, the component does not create the graphics world.

flags

Contains flags (see below) that determine some of the memory characteristics of the new graphics world.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

flags Constants

pixPurge

Make the base address for the offscreen pixel image purgeable.

noNewDevice

Do not create a new offscreen GDevice (IV–2264) structure; instead, use either the GDevice structure you specify or the GDevice structure for a video card on the user's system.

useTempMem

Create the base address for an offscreen pixel image in temporary memory. You generally should not use this flag.

keepLocal

Create a pixel image in main memory where it cannot be cached to a graphics accelerator card.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Creating a Compression Graphics World” (V–2804)

Related Java Methods

quicktime.std.qtcomponents.ImageCompressionDialog.newGWorld(),

quicktime.qd.QDGraphics.QDGraphics()

SCPositionDialog

Helps position a dialog box on the screen.

```
ComponentResult SCPositionDialog (
    ComponentInstance ci,
    short id,
    Point *where );
```

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from OpenDefaultComponent (II–1131).

id

The **resource** number of a 'DLOG' resource. The function positions the dialog box that corresponds to this resource.

SCPositionRect

where

A pointer to a `Point` (IV–2339) structure identifying the desired location of the upper-left corner of the dialog box in global coordinates. This parameter allows you to indicate how you want to position the dialog box on the screen.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Positioning Dialog Boxes and Rectangles” (V–2803)

SCPositionRect

Positions a rectangle on the screen.

```
ComponentResult SPositionRect (
    ComponentInstance ci,
    Rect *rp,
    Point *where );
```

ci

Identifies your application’s connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

rp

A pointer to a `Rect` (IV–2399) structure. When you call the function, this structure should contain the rectangle’s current global coordinates. The function adjusts the coordinates in the structure to reflect the rectangle’s new position.

where

A pointer to a `Point` (IV–2339) structure identifying the desired location of the upper-left corner of the rectangle in global coordinates. This parameter allows your application to position the rectangle on the screen.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Positioning Dialog Boxes and Rectangles” (V–2803)

SCRequestImageSettings

Displays the standard image dialog box to the user and shows default settings you have established.

```
ComponentResult SCRequestImageSettings (
    ComponentInstance ci );
```

ci

Identifies your application’s connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Use this function to retrieve the user’s preferences for compressing a single image; use `SCRequestSequenceSettings` (III–1556) when you are working with an image sequence. Both functions manipulate the compression settings that the component stores for you.

The component derives the current settings when you may supply an image to the component from which it can derive default settings. If you have not set any defaults, but you do supply a test image for the dialog, the component examines the test image and derives appropriate default values based upon its characteristics. If you have not set any defaults and do not supply a test image, the component uses its own default values.

Special Considerations

You may modify the settings by using `SCSetInfo` (III–1558). You may customize the dialog boxes by specifying a modal-dialog hook function or a custom button. You may use the custom button to invoke an ancillary dialog box that is specific to your application.

Version Notes

Introduced in QuickTime 3 or earlier.

SCRequestSequenceSettings

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Displaying the Standard Image-Compression Dialog Box” (V–2801), “Managing the Sound Dialog” (V–2895)

Related Java Methods

```
quicktime.util.QTHandleRef.fromCompressionDialogState(),  
quicktime.std.qtcomponents.ImageCompressionDialog.requestImageSettings(),  
quicktime.std.qtcomponents.CompressionDialog.requestSettings()
```

See Also

See `SCRequestSequenceSettings` (III–1556).

SCRequestSequenceSettings

Displays the standard sequence dialog box to the user and shows default settings you have established.

```
ComponentResult SCRequestSequenceSettings (  
    ComponentInstance ci );
```

ci

Identifies your application’s connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Use `SCRequestSequenceSettings` to retrieve the user’s preferences for compressing an image sequence; use `SCRequestImageSettings` (III–1555) when you are working with a single image. Both functions manipulate the compression settings that the component stores for you.

The component derives the current settings when you may supply an image to the component from which it can derive default settings. If you have not set any defaults, but you do supply a test image for the dialog, the component examines the test image and derives appropriate default values based upon its characteristics. If you have not set any defaults and do not supply a test image, the component uses its own default values.

Special Considerations

You may modify the settings by using `SCSetInfo` (III–1558). You may customize the dialog boxes by specifying a modal-dialog hook function or a custom button. You

may use the custom button to invoke an ancillary dialog box that is specific to your application.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Displaying the Standard Image-Compression Dialog Box” (V–2801)

Related Java Methods

`quicktime.std.qtcomponents.ImageCompressionDialog.requestSequenceSettings()`

See Also

See `SCRequestImageSettings` (III–1555).

SCSetCompressFlags

Sets compression flags for a standard image-compression dialog component.

```
ComponentResult SCSetCompressFlags (
    ComponentInstance ci,
    long flags );
```

ci

Identifies your application’s connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

flags

Flags (see below) to set.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`scCompressFlagIgnoreIdenticalFrames`
Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Related Java Methods

`quicktime.std.qtcomponents.ImageCompressionDialog.setCompressFlags()`

SCSetInfo

See Also

For the corresponding get function, see `SCGetCompressFlags` (III–1543).

SCSetInfo

Modifies the standard dialog component’s configuration information.

```
ComponentResult SCSetInfo (  
    ComponentInstance ci,  
    OSType           infoType,  
    void             *info );
```

ci

Identifies your application’s connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

infoType

A constant (see below) that specifies the type of information you want to set.

info

A pointer to a field that contains the new information. The `infoType` constant descriptions (see below) include information about this field.

function result See “Error Codes” (IV–2663). If the component cannot satisfy your request, it returns a result code of `scTypeNotFoundErr`. It returns `noErr` if there is no error.

infoType Constants

`scSpatialSettingsType`

The component’s spatial compression parameters in an `SCSpatialSettings` (IV–2423) structure.

`scTemporalSettingsType`

The component’s temporal compression parameters in an `SCTemporalSettings` (IV–2427) structure.

`scDataRateSettingsType`

Information about the component’s compression data rate in an `SCDataRateSettings` (IV–2417) structure.

`scColorTableType`

The component’s color table as a pointer to a `CTabHandle`, which references a `ColorTable` (IV–2210) structure.

scProgressProcType

A pointer to a `ICMPProgressProcRecord` (IV–2284) structure. Set the pointer to `NIL` to clear the current **progress function**; in this case, the standard dialog component does not report its progress to the user. Set the pointer to `-1` to use the component's default progress function.

scExtendedProcsType

A pointer to an `SCExtendedProcs` (IV–2418) structure.

scPreferenceFlagsType

A pointer to a long integer. It contains constants (see below) that define preference flags.

scSettingsStateType

A pointer to a handle to the component's complete configuration. Your application is responsible for disposing of the handle when you are done with it. Set the pointer to `NIL` to reset the component to its default configuration.

Your application is not concerned with the content of the configuration information that is returned. The standard dialog component saves its configuration in a format that it understands. This request affects only those settings that are valid across system restarts, such as the spatial and temporal compression parameters and the data-rate settings. When you retrieve the settings, the standard dialog component creates an appropriately-sized handle and places its current configuration information into the handle. When you modify the settings, you supply the configuration information in the handle, and the component copies the data out of this handle.

scSequenceIDType

The component's current image-compression sequence identifier being used by the component's `SCCompressSequenceFrame` (III–1534) function. You supply a pointer to a field of type `ImageSequence`; see “Codec Types” (IV–2619). The standard dialog component returns the current sequence identifier in that field.

scWindowPositionType

A pointer to a `Point` (IV–2339) structure. If you are retrieving this information, the standard dialog component places the coordinates of the upper-left corner of the dialog box into this structure; if you are changing the dialog box's position, place the new coordinates into the structure; the component uses those coordinates to position the dialog box.

Normally you should not need to use this request. By default, the standard dialog component centers the dialog box on the screen that is best-suited to

display your test image. The component also saves the last window position for movable modal dialogs.

scCodecFlagsType

A pointer to a field of type `CodecFlags` that contains flag constants (see below); see “Codec Types” (IV–2619). If you are retrieving the flags, the standard dialog component places the current flags into this field. If you are setting new flag values, place your desired settings into the field; the component uses these new flag settings.

scPreferenceFlagsType Constants

scListEveryCodec

Controls the contents of the pop-up menu of compressors. If you set this flag to 1, the standard image-compression dialog component lists every compressor component that is present in the system. Each entry in the list contains the name of a compressor component. The user may then select a specific component from the list. If you set this flag to 0, the list contains one entry for each type of compressor component that is present in the system. Each list entry contains the name of a compressor type (for example, a list entry might contain “Animation” for the animation compressor). The user may then select a type of compressor; it is your application’s responsibility to select an appropriate compressor of that type.

scAllowZeroFrameRate

Determines whether the component allows the user to specify a value of 0 for the frame rate. If you set this flag to 1, the component allows the user to specify either 0 or nothing for the frame rate. The component then includes a “best rate” entry in the pop-up menu. If the user specifies 0, the component sets the `frameRate` field in the `SCTemporalSettings` (IV–2427) structure to 0. Your application must then determine the best frame rate for the movie. If you set this flag to 0, the component does not allow the user to enter 0 for the frame rate. In this case, the user must select a specific frame rate.

scAllowZeroKeyFrameRate

Similar to the `scAllowZeroFrameRate` flag, this flag determines whether the component allows the user to specify a value of 0 for the **key frame** rate. If you set this flag to 1, the component allows the user to specify 0 for the frame rate. If the user specifies 0, the component sets the `keyFrameRate` field in the `SCTemporalSettings` (IV–2427) structure to 0. Your application must then determine the best key frame rate for the movie. If you set this flag to 0, the component does not allow the user to specify 0 for the frame rate. In this case, if the user has enabled temporal compression by checking the key frame checkbox, the user must also select a specific key frame rate.

scShowBestDepth

Determines whether the component includes a “best depth” entry in the pop-up menu for pixel depth. If you set this flag to 1, the component includes a “best depth” entry in the pop-up menu. If the user selects “best depth,” the component sets the depth to 0. Your application must then determine the best pixel depth for the movie. If you set this flag to 0, the component does not include a “best depth” entry in the pop-up menu. The user must select a depth from among the other available choices.

scUseMovableModal

Determines whether the standard compression dialog is a movable or a stationary dialog. Set this flag to 1 to create a movable dialog. In this case, you should provide an event filter function to handle update events; use the `scExtendedProcsType` request (see above).

scCodecFlagsType Constants**codecFlagUseImageBuffer**

Controls whether the decompressor allocates an offscreen buffer for decompression. If your application sets this flag to 1, the decompressor allocates an offscreen buffer the size of the compressed image. If you set this flag to 0, the decompressor does not use an offscreen image buffer. These image buffers are useful when decompressing sequences that were created using temporal compression.

codecFlagUseScreenBuffer

Controls whether the decompressor allocates an offscreen destination buffer during decompression. If you set this flag to 1, the decompressor allocates an offscreen buffer the size of the destination screen. If you set this flag to 0, the decompressor does not use an offscreen screen buffer. Using a screen buffer helps to reduce tearing that may result when decompressing directly to the screen.

codecFlagUpdatePrevious

Controls whether the compressor updates the previous image buffer during compression. This flag is only used with sequences that are being temporally compressed. If you set this flag to 1, the compressor copies the current source image into the previous frame buffer at the end of the frame compression.

codecFlagNoScreenUpdate

Controls whether the decompressor updates the screen image. If you set this flag to 1, the decompressor does not write the current frame to the screen, but does write the frame to its offscreen image buffer (if one was allocated). If you set this flag to 0, the decompressor writes the frame to the screen.

`codecFlagWasCompressed`

Indicates to the compressor that the image to be compressed has been compressed before. This information may be useful to compressors that can compensate for the image degradation that may otherwise result from repeated compression and decompression of the same image. Set this flag to 1 to indicate that the image was previously compressed. Set this flag to 0 if the image was not previously compressed.

`codecFlagDontOffscreen`

Controls whether the decompressor uses the offscreen buffer during sequence decompression. This flag is only used with sequences that have been temporally compressed. If this flag is set to 1, the decompressor does not use the offscreen buffer during decompression. Instead, the decompressor returns an error. This allows your application to refill the offscreen buffer. If this flag is set to 0, the decompressor uses the offscreen buffer if appropriate.

`codecFlagUpdatePreviousComp`

Controls whether the compressor updates the previous image buffer with the decompressed image data. This flag is only used with temporal compression and is similar to the `codecFlagUpdatePrevious` flag. As with the `codecFlagUpdatePrevious` flag, if you set this flag to 1, the compressor updates the previous frame buffer at the end of the frame compression. However, this flag causes the Image Compression Manager to update the frame buffer using an image obtained by decompressing the results of the most recent compression operation, rather than the source image.

`codecFlagForceKeyFrame`

Controls whether the compressor creates a **key frame** from the current image. This flag is only used with temporal compression. If you set this flag to 1, the compressor makes the current image a key frame. If you set this flag to 0, the compressor decides based on other criteria, such as the key frame rate, whether to create a key frame from the current image.

`codecFlagOnlyScreenUpdate`

Controls whether the decompressor decompresses the current frame. If you set this flag to 1, the decompressor writes the contents of its offscreen image buffer to the screen, but does not decompress the current frame. If you set this flag to 0, the decompressor decompresses the current frame and writes it to the screen. You can set this flag to 1 only if you have allocated an offscreen image buffer for use by the decompressor.

`codecFlagLiveGrab`

Indicates to the compressor whether the current sequence results from grabbing live video. When working with live video, compressors operate as

quickly as possible and disable some additional processing, such as compensation for previously compressed data. Set this flag to 1 when you are compressing from a live video source; the compressor then operates as quickly as it can.

`codecFlagUsedNewImageBuffer`

Indicates to your application that the decompressor used the offscreen image buffer for the first time when it processed this frame. If this flag is set to 1, the decompressor used the image buffer for this frame and this is the first time the decompressor used the image buffer in this sequence. If this flag is set to 0, the decompressor did not use the image buffer.

`codecFlagUsedImageBuffer`

Indicates to your application that the decompressor used the offscreen image buffer for this frame. If this flag is set to 1, the decompressor used the image buffer. If this flag is set to 0, the decompressor did not use the image buffer.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing the Sound Dialog” (V–2895), “Working With Image or Sequence Settings” (V–2802)

Related Java Methods

```
quicktime.std.qtcomponents.ImageCompressionDialog.setInfoSpatialSettings()
quicktime.std.qtcomponents.ImageCompressionDialog.setInfoTemporalSettings()
quicktime.std.qtcomponents.ImageCompressionDialog.setInfoDataRateSettings()
quicktime.std.qtcomponents.ImageCompressionDialog.setInfoColorTable()
quicktime.std.qtcomponents.CompressionDialog.setInfoState()
quicktime.std.qtcomponents.CompressionDialog.setInfoPreferences()
quicktime.std.qtcomponents.SoundCompressionDialog.setInfoSampleRate()
quicktime.std.qtcomponents.SoundCompressionDialog.setInfoSampleSize()
quicktime.std.qtcomponents.SoundCompressionDialog.setInfoChannelCount()
quicktime.std.qtcomponents.SoundCompressionDialog.setInfoCompression()
quicktime.std.qtcomponents.SoundCompressionDialog.setInfoCompressionList()
```

See Also

For the corresponding get function, see `SCGetInfo` (III–1545).

SCSetSettingsFromAtomContainer

SCSetSettingsFromAtomContainer

Sets the standard image-compression component's current configuration from data in a QT atom container.

```
ComponentResult SCSetSettingsFromAtomContainer (
    ComponentInstance    ci,
    QTAtomContainer      settings );
```

ci

Standard compression component instance.

settings

A QT atom container reference to the settings.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The settings QT atom container may contain atoms other than those expected by the particular component type or may be missing certain atoms. The function will only use settings it understands.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Related Java Methods

quicktime.std.qtcomponents.CompressionDialog.setInfoState()

SCSetTestImagePictFile

Sets the dialog box's test image from a Picture (IV–2331) structure that is stored in a **picture file**.

```
ComponentResult SCSetTestImagePictFile (
    ComponentInstance    ci,
    short                testFileRef,
    Rect                 *testRect,
    short                testFlags );
```

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II-1131).

testFileRef

Identifies the file that contains the new test image. Your application is responsible for opening this file before calling this function. You must also close the file when you are done with it. You must clear the image or close your connection to the standard image-compression dialog component before you close the file. If the file contains a large image, the component may take some time to display the standard image-compression dialog box. In this case, the component displays the watch cursor while it loads the test image.

testRect

A pointer to a `Rect` (IV-2399) structure. This rectangle specifies, in the coordinate system of the source image, the area of interest or point of interest in the test image. The area of interest defines a portion of the test image that is to be shown to the user in the dialog box. Use this parameter to direct the component to a specific portion of the test image. The component uses the value of the `testFlags` parameter to determine how it transforms large images before displaying them to the user.

testFlags

Constants (see below) that specify how the component is to display a test image that is larger than the test image portion of the dialog box. If you set this parameter to 0, the component uses a default method of its own choosing. In all cases, the component centers the area or point of interest in the test image portion of the dialog box, and then displays some part of the test image.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

testFlags Constants

scPreferCropping

The component should crop the test image to fit the test image portion of the dialog box. The component displays the part of the image that fits in the test image portion of the box. If the image is smaller than the space allotted in the dialog box, the component does not alter the image before displaying it; the resulting image is smaller than the available space.

scPreferScaling

The component should scale the test image to fit the test image portion of the dialog box, by shrinking it.

SCSetTestImagePictHandle

scPreferScalingAndCropping

The component should both scale and crop the test image. This option is useful with very large test images. The component first shrinks the image to approximately the size of the test image portion of the dialog box, then trims the image so that it fits the available space.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Specifying a Test Image” (V–2803)

Related Java Methods

`quicktime.std.qtcomponents.ImageCompressionDialog.setTestImagePictFile()`

SCSetTestImagePictHandle

Sets the dialog box’s test image from a `Picture` (IV–2331) structure that is stored in a handle.

```
ComponentResult SCSetTestImagePictHandle (
    ComponentInstance ci,
    PicHandle         testPict,
    Rect              *testRect,
    short             testFlags );
```

`ci`

Identifies your application’s connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II–1131).

`testPict`

Identifies a handle that contains the new test image. Your application is responsible for disposing of this handle when you are done with it. You must clear the image or close your connection to the standard image-compression dialog component before you dispose of this handle or close the corresponding **resource** file. You must set this handle as nonpurgeable.

`testRect`

A pointer to a `Rect` (IV–2399) structure. This structure specifies, in the coordinate system of the source image, the area of interest or point of interest in the test image. The area of interest defines a portion of the test image that is to be shown to the user in the dialog box. Use this parameter to direct the

component to a specific portion of the test image. The component uses the value of the `testFlags` parameter to determine how it transforms this image before displaying it to the user. The component uses the `testFlags` parameter only when the test image is larger than the test image portion of the dialog box.

`testFlags`

Constants (see below) that specify how the component is to display a test image that is larger than the test image portion of the dialog box. If you set this parameter to 0, the component uses a default method of its own choosing. In all cases, the component centers the area or point of interest in the test image portion of the dialog box, and then displays some part of the test image.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

testFlags Constants

`scPreferCropping`

The component should crop the test image to fit the test image portion of the dialog box. The component displays the part of the image that fits in the test image portion of the box. If the image is smaller than the space allotted in the dialog box, the component does not alter the image before displaying it; the resulting image is smaller than the available space.

`scPreferScaling`

The component should scale the test image to fit the test image portion of the dialog box, by shrinking it.

`scPreferScalingAndCropping`

The component should both scale and crop the test image. This option is useful with very large test images. The component first shrinks the image to approximately the size of the test image portion of the dialog box, then trims the image so that it fits the available space.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Specifying a Test Image” (V–2803)

Related Java Methods

`quicktime.std.qtcomponents.ImageCompressionDialog.setTestImagePict()`

SCSetTestImagePixMap

Sets the dialog box's test image from a `Picture` (IV-2331) structure that is stored in a `PixMap` (IV-2332) structure.

```
ComponentResult SCSetTestImagePixMap (  
    ComponentInstance    ci,  
    PixMapHandle         testPixMap,  
    Rect                 *testRect,  
    short                testFlags );
```

`ci`

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent` (II-1131).

`testPixMap`

A handle to a `PixMap` (IV-2332) structure that contains the new test image. Your application is responsible for creating this structure before calling the function. You must also dispose of the structure when you are done with it. You must clear the image or close your connection to the standard image-compression dialog component before you dispose of the structure.

`testRect`

A pointer to a `Rect` (IV-2399) structure. This rectangle specifies, in the coordinate system of the source image, the area of interest or point of interest in the test image. The area of interest defines a portion of the test image that is to be shown to the user in the dialog box. Use this parameter to direct the component to a specific portion of the test image. The component uses the value of the `testFlags` parameter to determine how it transforms large images before displaying them to the user.

`testFlags`

Constants (see below) that specify how the component is to display a test image that is larger than the test image portion of the dialog box. If you set this parameter to 0, the component uses a default method of its own choosing. In all cases, the component centers the area or point of interest in the test image portion of the dialog box, and then displays some part of the test image.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

testFlags Constants

`scPreferCropping`

The component should crop the test image to fit the test image portion of the dialog box. The component displays the part of the image that fits in the test

image portion of the box. If the image is smaller than the space allotted in the dialog box, the component does not alter the image before displaying it; the resulting image is smaller than the available space.

`scPreferScaling`

The component should scale the test image to fit the test image portion of the dialog box, by shrinking it.

`scPreferScalingAndCropping`

The component should both scale and crop the test image. This option is useful with very large test images. The component first shrinks the image to approximately the size of the test image portion of the dialog box, then trims the image so that it fits the available space.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Specifying a Test Image” (V–2803)

Related Java Methods

`quicktime.std.qtcomponents.ImageCompressionDialog.setTestImagePixmap()`

SelectMovieAlternates

Instructs the Movie Toolbox to select appropriate tracks immediately.

```
void SelectMovieAlternates (
    Movie    theMovie );
```

`theMovie`

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Alternate Tracks” (V–2738)

SetAutoTrackAlternatesEnabled

Related Java Methods

```
quicktime.std.movies.Movie.selectAlternates()
```

SetAutoTrackAlternatesEnabled

Enables or disables automatic track selection by the Movie Toolbox.

```
void SetAutoTrackAlternatesEnabled (  
    Movie      theMovie,  
    Boolean    enable );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

enable

Controls automatic track selection. Set this parameter to `TRUE` to enable automatic track selection. Set this parameter to `FALSE` to disable automatic track selection.

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Alternate Tracks” (V–2738)

Related Java Methods

```
quicktime.std.movies.Movie.setAutoTrackAlternatesEnabled()
```

SetComponentInstanceA5

Obsolete.

```
void SetComponentInstanceA5 (  
    ComponentInstance  aComponentInstance,  
    long               theA5 );
```


Version Notes

Introduced in QuickTime 3 or earlier. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: `Components.h`

See Also

For the corresponding get function, see `GetComponentInstanceA5` (I-390).

SetComponentInstanceError

Lets a component pass error information to the Component Manager other than through its return value.

```
void SetComponentInstanceError (
    ComponentInstance  aComponentInstance,
    OSErr              theError );
```

`aComponentInstance`

A component instance. Your software obtains this reference from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131). The Component Manager also provides a component instance to your component as the first parameter in the `params` field of the `ComponentParameters` (IV-2216) record.

`theError`

The new value for the current error. The Component Manager uses this value to set the current error for the connection specified by the `aComponentInstance` parameter.

Discussion

Although your component usually returns error information as its function result, your component can choose to use this function to pass error information to the Component Manager. The Component Manager uses this error information to set the current error value for the appropriate connection. Applications can then retrieve this error information by calling `GetComponentInstanceError` (I-390).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V-2782)

SetComponentInstanceStorage

See Also

For the corresponding get function, see GetComponentInstanceError (I–390).

SetComponentInstanceStorage

Lets a component pass the Component Manager a handle to the memory that was allocated for the connection to it.

```
void SetComponentInstanceStorage (
    ComponentInstance    aComponentInstance,
    Handle                theStorage );
```

aComponentInstance

A component instance. Your software obtains this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131). The Component Manager also provides a component instance to your component as the first parameter in the params field of the ComponentParameters (IV–2216) record.

theStorage

A handle to the memory that your component has allocated for the connection. Your component must allocate this memory in the current heap. The Component Manager saves this handle and provides it to your component, along with other parameters, in subsequent requests to this connection.

Special Considerations

Your component should dispose of any allocated memory for the connection only in response to the close request. Note that whenever an open request fails, the Component Manager always issues the close request. Furthermore, the value stored by this function is always passed to the close request, so it must be valid or NIL. If the open request tries to dispose of its allocated memory before returning, it should call this function again with a NIL handle to keep the Component Manager from passing an invalid handle to the close request.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Components.h

Programming summary: “Functions Used By Components” (V–2782)

See Also

For the corresponding get function, see GetComponentInstanceStorage (I–391).

SetComponentRefcon

Sets the reference constant for a component.

```
void SetComponentRefcon (
    Component    aComponent,
    long         theRefcon );
```

aComponent

A component instance. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131). The Component Manager also provides a component instance to your component as the first parameter in the `params` field of the `ComponentParameters` (IV–2216) record.

theRefcon

The reference constant value that you want to set for your component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

See Also

For the corresponding get function, see `GetComponentRefcon` (I–394).

SetCompressedPixMapInfo

Stores information about a compressed image for `StdPix` (III–1912).

```
OSErr SetCompressedPixMapInfo (
    PixMapPtr          pix,
    ImageDescriptionHandle desc,
    Ptr                data,
    long               bufferSize,
    ICMDataProcRecordPtr dataProc,
    ICMProgressProcRecordPtr progressProc );
```

pix

A pointer to a `PixMap` (IV–2332) structure that holds compressed image data.

desc

A handle to the `ImageDescription` (IV–2285) structure that defines the compressed image.

SetCSequenceDataRateParams

data

A pointer to the buffer for the compressed image data. If the entire compressed image cannot be stored at this location, you may assign a data-loading function (see the *dataProc* parameter, below). This pointer must contain a **32-bit clean** address.

bufferSize

The size of the buffer to be used by the data-loading function specified by the *dataProc* parameter. If there is no data-loading function defined for this operation, set this parameter to 0.

dataProc

A pointer to an *ICMDataProcRecord* (IV–2279) structure. If there is not enough memory to store the compressed image, the decompressor calls an *ICMDataProc* (III–2090) callback that you provide, which loads more compressed data. If you do not want to assign a data-loading function, set this parameter to *NIL*.

progressProc

A pointer to an *ICMProgressProcRecord* (IV–2284) structure. During the decompression operation, the decompressor may occasionally call an *ICMProgressProc* (III–2093) callback that you provide, in order to report its progress. If you do not want to assign a **progress function**, set this parameter to *NIL*. If you pass a value of –1, you obtain a standard progress function.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: *ImageCompression.h*

Programming summary: “Working With the *StdPix* Function” (V–2797)

See Also

For the corresponding get function, see *GetCompressedPixelFormatInfo* (I–397).

SetCSequenceDataRateParams

Communicates information to compressors that can constrain compressed data in a particular sequence to a specific data rate.

```
OSErr SetCSequenceDataRateParams (
    ImageSequence      seqID,
    DataRateParamsPtr  params );
```

seqID

The unique sequence identifier assigned by CompressSequenceBegin (I–119) or DecompressSequenceBegin (I–238).

params

A pointer to a DataRateParams (IV–2231) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Constraining Compressed Data” (V–2795)

Related Java Methods

quicktime.std.image.CSequence.setDataRateParams()

See Also

For the corresponding get function, see GetCSequenceDataRateParams (I–403).

SetCSequenceFlushProc

Assigns a data-unloading function to a sequence.

```
OSErr SetCSequenceFlushProc (
    ImageSequence      seqID,
    ICMFlushProcRecordPtr flushProc,
    long               bufferSize );
```

seqID

The unique sequence identifier assigned by CompressSequenceBegin (I–119) or DecompressSequenceBegin (I–238).

flushProc

A pointer to an ICMFlushProcRecord (IV–2280) structure. If there is not enough memory to store the compressed image, the compressor calls an ICMFlushProc (III–2091) callback that you provide, which unloads some of the compressed data. If you have not provided such a data-unloading function, set this parameter to NIL. In this case, the compressor writes the entire compressed image into the memory location specified by the data parameter to CompressSequenceFrame (I–124).

SetCSequenceFrameNumber

bufferSize

The size of the buffer to be used by the data-unloading function specified by the `flushProc` parameter. If you have not specified such a data-unloading function, set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Data-unloading functions allow compressors to work with images that cannot fit in memory. During the compression operation, the compressor calls the data-unloading function whenever it has accumulated a specified amount of compressed data. Your data-unloading function then writes the compressed data to some other device, freeing buffer space for more compressed data. The compressor starts using the data-unloading function with the next image in the sequence.

Special Considerations

There is no parameter to the `CompressSequenceBegin` (I–119) function that allows you to assign a data-unloading function to a sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Compression Parameters” (V–2794)

SetCSequenceFrameNumber

Notifies the compressor in use for the specified sequence that frames are being compressed out of order.

```
OSErr SetCSequenceFrameNumber (
    ImageSequence    seqID,
    long             frameNumber );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

frameNumber

The frame number of the frame that is being compressed out of sequence.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This information is necessary only for compressors that are sequence-sensitive.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Related Java Methods

`quicktime.std.image.CSequence.setFrameNumber()`

See Also

For the corresponding get function, see `GetCSequenceFrameNumber` (I-404).

S

SetCSequenceKeyFrameRate

Adjusts the **key frame** rate for the current sequence.

```
OSErr SetCSequenceKeyFrameRate (
    ImageSequence  seqID,
    long           keyFrameRate );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I-119) or `DecompressSequenceBegin` (I-238).

keyFrameRate

The maximum number of frames allowed between **key frames**. Set this parameter to 1 to specify all key frames, to 2 to specify every other frame as a key frame, to 3 to specify every third frame as a key frame, and so forth. The compressor determines the optimum placement for key frames based upon the amount of redundancy between adjacent images in the sequence.

Consequently, the compressor may insert key frames more frequently than you have requested. However, the compressor will never place fewer key frames than is indicated by this parameter. If you set this parameter to 0, the Image Compression Manager only places key frames in the compressed sequence when you call `CompressSequenceFrame` (I-124), setting the `codecFlagForceKeyFrame` flag in the `flags` parameter. The compressor ignores this parameter if you have not requested temporal compression; that is, you have passed 0 for the `temporalQuality` parameter of `CompressSequenceBegin` (I-119).

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

SetCSequencePreferredPacketSize

Discussion

Key frames provide points from which a temporally compressed sequence may be decompressed. Use this parameter to control the frequency at which the compressor places **key frames** into the compressed sequence. The new key frame rate takes effect with the next image in the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Compression Parameters” (V–2794)

Related Java Methods

`quicktime.std.image.CSequence.setKeyFrameRate()`

See Also

For the corresponding get function, see `GetCSequenceKeyFrameRate` (I–405).

SetCSequencePreferredPacketSize

Sets the preferred packet size for a sequence.

```
OSErr SetCSequencePreferredPacketSize (
    ImageSequence    seqID,
    long             preferredPacketSizeInBytes );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

preferredPacketSizeInBytes

The preferred packet size in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

This function was added in QuickTime 2.5 to support video conferencing applications by making each transmitted packet an independently decodable chunk of data.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Compression Parameters” (V–2794)

Related Java Methods

`quicktime.std.image.CSequence.setPreferredPacketSize()`

SetCSequencePrev

Allows the application to set the pixel map and boundary rectangle used by the previous frame in temporal compression.

```
OSErr SetCSequencePrev (
    ImageSequence    seqID,
    PixMapHandle     prev,
    const Rect       *prevRect );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

prev

A handle to the new previous image buffer. You must allocate this buffer using the same pixel depth and `ColorTable` (IV–2210) structure as the source image buffer that you specified with the `src` parameter when you called `CompressSequenceBegin` (I–119). The compressor uses this buffer to store a previous image against which the current image is compared when performing temporal compression. The compressor manages the contents of this buffer based upon several considerations, such as the **key frame** rate and the degree of difference between compared images. The current image is stored in the buffer referred to by the `src` parameter to `CompressSequenceBegin`.

prevRect

A pointer to a `Rect` (IV–2399) structure that defines the portion of the previous image to use for temporal compression. The compressor uses this portion of the previous image as the basis of comparison with the current image. This rectangle must be the same size as the source rectangle you specify with the `srcRect` parameter to `CompressSequenceBegin` (I–119). To get the boundary of a source pixel map, set this parameter to `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

When you start compressing a sequence, you may assign a previous frame buffer and rectangle with the `prev` and `prevRect` parameters to `CompressSequenceBegin` (I–119). If you specified a `NIL` value for the `prev` parameter, the compressor allocates an offscreen buffer for the previous frame. In either case you may use this function to assign a new previous frame buffer.

Special Considerations

This is a very specialized function; your application should not need to call it under most circumstances.

SetCSequenceQuality

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Compression Parameters” (V–2794)

Related Java Methods

`quicktime.std.image.CSequence.setPrev()`

SetCSequenceQuality

Adjusts the spatial or temporal quality for the current sequence.

```
OSErr SetCSequenceQuality (
    ImageSequence    seqID,
    CodecQ           spatialQuality,
    CodecQ           temporalQuality );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

spatialQuality

A constant (see below) that specifies the desired compressed image quality.

temporalQuality

A constant (see below) that specifies the desired sequence temporal quality.

This parameter governs the level of compression you desire with respect to information between successive frames in the sequence. Set this parameter to 0 to prevent the compressor from applying temporal compression to the sequence.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

spatialQuality and temporalQuality Constants

codecMinQuality

The minimum valid value for a `CodecQ` field.

codecLowQuality

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

codecNormalQuality

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

You originally set the default spatial and temporal quality values for a sequence with `CompressSequenceBegin` (I–119). The new quality parameters take effect with the next frame in the sequence.

Special Considerations

If you change the quality settings while processing an image sequence, you affect the maximum image size that you may receive during sequence compression. Consequently, you should call `GetMaxCompressionSize` (I–420) after you change the quality settings. If the maximum size has increased, you should reallocate your image buffers to accommodate the larger image size.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Compression Parameters” (V–2794)

Related Java Methods

`quicktime.std.image.CSequence.setQuality()`

SetDefaultComponent

Changes the search order for registered components.

```
OSErr SetDefaultComponent (
    Component      aComponent,
    short          flags );
```

`aComponent`

The component that is to be placed at the front of the search chain. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131). The Component Manager also provides a

SetDefaultOutputVolume

component instance to your component as the first parameter in the `params` field of the `ComponentParameters` (IV–2216) record.

`flags`

A constant (see below) that controls which fields of the `ComponentDescription` (IV–2212) structure the Component Manager examines during the reorder operation. Set the appropriate flags to 1 to define the fields that are examined.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constants

`defaultComponentIdentical`

The Component Manager places the specified component in front of all other components that have the same component description.

`defaultComponentAnyFlags`

The Component Manager ignores the value of the `componentFlags` field during the reorder operation.

`defaultComponentAnyManufacturer`

The Component Manager ignores the value of the `componentManufacturer` field during the reorder operation.

`defaultComponentAnySubType`

The Component Manager ignores the value of the `componentSubType` field during the reorder operation.

Discussion

You specify a component that is to be placed at the front of the search chain, along with control information that governs the reordering operation. The order of the search chain influences which component the Component Manager selects in response to an application’s use of `OpenDefaultComponent` (II–1131) and `FindNextComponent` (I–360).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

SetDefaultOutputVolume

Sets the default volume of the sound output device.

`OSErr SetDefaultOutputVolume (`

```
long    level );
```

level

The desired default volume level. The values you can specify in the high and low words of the *level* parameter range from 0 (silence) to 0x0100 (full volume).

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding get function, see `GetDefaultOutputVolume` (I–408).

SetDSequenceAccuracy

Adjusts the decompression accuracy for the current sequence.

```
OSErr SetDSequenceAccuracy (
    ImageSequence    seqID,
    CodecQ           accuracy );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

accuracy

A constant (see below) that specifies the accuracy desired in the decompressed image.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

accuracy Constants

codecMinQuality

The minimum valid value for a `CodecQ` field.

codecLowQuality

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

SetDSequenceDataProc

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a CodecQ field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

The accuracy parameter governs how precisely the decompressor decompresses the image data. Some decompressors may choose to ignore some image data to improve decompression speed. A new accuracy value takes effect with the next frame in the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Decompression Parameters” (V-2795)

Related Java Methods

`quicktime.std.image.DSequence.setAccuracy()`

See Also

You set the default accuracy value for a sequence with the accuracy parameter to `DecompressSequenceBegin` (I-238).

SetDSequenceDataProc

Assigns a data-loading function to a sequence.

```
OSErr SetDSequenceDataProc (
    ImageSequence      seqID,
    ICMDataProcRecordPtr dataProc,
    long               bufferSize );
```

seqID

The unique sequence identifier assigned by CompressSequenceBegin (I–119) or DecompressSequenceBegin (I–238).

dataProc

A pointer to an ICMDataProcRecord (IV–2279) structure. If the data stream is not all in memory when your program calls DecompressSequenceFrame (I–242), the decompressor calls an ICMDataProc (III–2090) callback that you provide, which loads more compressed data. If you have not provided such a data-loading function, or if you want the decompressor to stop using your data-loading function, set this parameter to NIL. In this case, the entire image must be in memory at the location specified by the data parameter to DecompressSequenceFrame.

bufferSize

The size of the buffer to be used by the data-loading function specified by the dataProc parameter. If you have not specified a data-loading function, set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Data-loading functions allow decompressors to work with images that cannot fit in memory. During the decompression operation the decompressor calls the data-loading function whenever it has exhausted its supply of compressed data.

Special Considerations

There is no parameter to the DecompressSequenceBegin (I–238) function that allows you to assign a data-loading function to a sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Changing Sequence-Decompression Parameters” (V–2795)

SetDSequenceFlags

Sets data loading flags.

```
OSErr SetDSequenceFlags (
    ImageSequence  seqID,
    long           flags,
```

SetDSequenceMask

```
long flagsMask );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

flags

Flags (see below) for data loading.

flagsMask

Use this field to preserve the state of any flags you do not wish to alter. If a flag (see below) is set in this field, and is not set in the `flags` parameter, it will not be changed from its current setting.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constant

`codecDSequenceSingleField`

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `ImageCompression.h`

SetDSequenceMask

Assigns a clipping region to a sequence.

```
OSErr SetDSequenceMask (
    ImageSequence seqID,
    RgnHandle      mask );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

mask

A handle to a clipping region in the destination coordinate system. If specified, the decompressor applies this mask to the destination image. If you want to stop masking, set this parameter to `NIL`. The new region takes effect with the next frame in the sequence.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The decompressor draws only that portion of the decompressed image that lies within the specified clipping region. You should not dispose of this region until the Image Compression Manager is finished with the sequence, or until you set the mask either to NIL or to a different region by calling this function again.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Changing Sequence-Decompression Parameters” (V-2795)

Related Java Methods

`quicktime.std.image.DSequence.setMask()`

SetDSequenceMatrix

Assigns a mapping matrix to a sequence.

```
OSErr SetDSequenceMatrix (
    ImageSequence      seqID,
    MatrixRecordPtr    matrix );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I-119) or `DecompressSequenceBegin` (I-238).

matrix

A `MatrixRecord` (IV-2304) structure that specifies how to transform the image during decompression. You can use this structure to translate or scale the image during decompression. To set the matrix to identity, pass NIL in this parameter. The new matrix takes effect with the next frame in the sequence.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

The decompressor uses the matrix to create special effects with the decompressed image, such as translating or scaling the image.

Version Notes

Introduced in QuickTime 3 or earlier.

SetDSequenceMatte

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Decompression Parameters”
(V–2795)

Related Java Methods

`quicktime.app.image.ImagePresenter.setMatrix()`,
`quicktime.app.image.QTImageDrawer.setMatrix()`,
`quicktime.std.image.DSequence.setMatrix()`

See Also

For the corresponding get function, see `GetDSequenceMatrix` (I–410).

SetDSequenceMatte

Assigns a blend matte to a sequence.

```
OSErr SetDSequenceMatte (
    ImageSequence    seqID,
    PixMapHandle     matte,
    const Rect       *matteRect );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

matte

A handle to a `PixMap` (IV–2332) structure that contains a blend matte. You can use the blend matte to cause the decompressed image to be blended into the destination pixel map. The matte can be defined at any supported pixel depth; the matte depth need not correspond to the source or destination depths.

However, the matte must be in the coordinate system of the source image. If you want to turn off the blend matte, set this parameter to `NIL`.

matteRect

A pointer to a `Rect` (IV–2399) structure that defines the boundary rectangle for the matte. The decompressor uses only that portion of the matte that lies within the specified rectangle. This rectangle must be the same size as the source rectangle you specify with `SetDSequenceSrcRect` (III–1589) or with the `srcRect` parameter to `DecompressSequenceBegin` (I–238). To use the matte’s `PixMap` (IV–2332) structure bounds as the boundary rectangle, pass `NIL` in this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The decompressor uses the matte to blend the decompressed image into the destination pixel map. The new matte and matte boundary rectangle take effect with the next frame in the sequence. You should not dispose of the matte until the Image Compression Manager has finished with the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Decompression Parameters” (V–2795)

Related Java Methods

`quicktime.std.image.DSequence.setMatte()`

SetDSequenceSrcRect

Defines the portion of an image to decompress.

```
OSErr SetDSequenceSrcRect (
    ImageSequence  seqID,
    const Rect      *srcRect );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

srcRect

A pointer to a `Rect` (IV–2399) structure that defines the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the compressed image, which is defined by (0,0) and `((**desc).width(**desc).height)`, where `desc` refers to the `ImageDescription` (IV–2285) structure you supply to `DecompressSequenceBegin` (I–238). If the `srcRect` parameter is `NIL`, the rectangle is set to the `Rect` structure in the `ImageDescription`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The decompressor acts on that portion of the compressed image that lies within this rectangle. A new source rectangle takes effect with the next frame in the sequence.

SetDSequenceTimeCode

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Changing Sequence-Decompression Parameters” (V-2795)

Related Java Methods

`quicktime.std.image.DSequence.setSrcRect()`

SetDSequenceTimeCode

Sets the timecode value for a frame that is about to be decompressed.

```
OSErr SetDSequenceTimeCode (
    ImageSequence    seqID,
    void             *timeCodeFormat,
    void             *timeCodeTime );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I-119) or `DecompressSequenceBegin` (I-238).

timeCodeFormat

A pointer to a `TimeCodeDef` (IV-2482) structure. You provide the appropriate timecode definition information for the next frame to be decompressed.

timeCodeTime

A pointer to a `TimeCodeRecord` (IV-2484) structure. You provide the appropriate time value for the next frame in the current sequence.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

QuickTime’s video media handler uses this function to set the timecode information for a movie. When a movie that contains timecode information starts playing, the media handler calls this function as it processes the movie’s first frame. The Image Compression Manager passes the timecode information straight through to the image decompressor component. That is, the Image Compression Manager does not make a copy of any of this timecode information. As a result, you must make sure that the data referred to by the `timeCodeFormat` and `timeCodeTime` parameters is valid until the next decompression operation completes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Decompression Parameters” (V–2795)

SetDSequenceTransferMode

Sets the mode used when drawing a decompressed image.

```
OSErr SetDSequenceTransferMode (
    ImageSequence    seqID,
    short            mode,
    const RGBColor   *opColor );
```

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

mode

A constant (see below) that specifies the transfer mode to be used when drawing the decompressed image. See also “Graphics Transfer Modes” (IV–2670).

opColor

Contains a pointer to the color for use in `addPin`, `subPin`, `blend`, and `transparent` operations. The Image Compression Manager passes this color to `QuickDraw` as appropriate. If `NIL`, the `opcolor` is left unchanged.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

mode Constants

blend

Replace the destination with a blend of the source and destination colors.

addPin

Replace the destination with the sum of the source and destination, up to a maximum value.

addOver

Replace the destination with the sum of the source and destination, but if the resulting red, green, or blue value exceeds 65536, then subtract 65536 from it.

SetDSequenceTransferMode

subPin

Replace the destination with the difference between the source and destination, but not less than a minimum value.

adMax

Compare the source and destination, and replace the destination with the greater value of each of the red, green, and blue components.

subOver

Replace the destination with the difference between the source and destination, but if the resulting red, green, or blue value is negative, then add 65536 to it.

adMin

Compare the source and destination, and replace the destination with the lesser value of each of the red, green, and blue components.

ditherCopy

Replace the destination with a **dither** mix of the source and destination.

transparent

Replace the destination with the source if the source is not equal to the background.

Discussion

For any given sequence, the default `opColor` value is 50 percent gray and the default mode is **ditherCopy**. The new mode takes effect with the next frame in the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Changing Sequence-Decompression Parameters” (V-2795)

Related Java Methods

```
quicktime.std.image.DSequence.setGraphicsMode(),  
quicktime.std.image.DSequence.setTransferMode()
```

See Also

The Image Compression Manager supports the same transfer modes that are supported by the Mac OS `CopyBits` routine; see *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

SetIdentityMatrix

Sets the contents of a matrix so that it performs no transformation.

```
void SetIdentityMatrix (
    MatrixRecord *matrix );
```

matrix

A pointer to a MatrixRecord (IV–2304) structure. The function updates the contents of this matrix so that the matrix describes the identity matrix.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Managing Matrices” (V–2764)

Related Java Methods

quicktime.std.image.Matrix.setIdentity()

SetImageDescriptionCTable

Updates the custom ColorTable (IV–2210) structure for an image.

```
OSErr SetImageDescriptionCTable (
    ImageDescriptionHandle desc,
    CTabHandle ctable );
```

desc

Contains a handle to the appropriate ImageDescription (IV–2285) structure. The function updates the size of the structure to accommodate the new ColorTable (IV–2210) structure and removes the old color table, if one is present.

ctable

A handle to the new ColorTable (IV–2210) structure. The function loads this color table into the ImageDescription (IV–2285) structure referred to by the *desc* parameter. Set this parameter to NIL to remove a ColorTable structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function does not change the image data, just the color table.

SetMediaDataHandler

Special Considerations

This function is rarely used. Typically, you supply the color table when your application compresses an image, and the Image Compression Manager stores the `ColorTable` (IV–2210) structure with the image.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pixel Maps” (V–2789)

Related Java Methods

`quicktime.std.image.ImageDescription.setCTable()`

See Also

For the corresponding get function, see `GetImageDescriptionCTable` (I–417).

SetMediaDataHandler

Assigns a data handler to a media.

```
OSErr SetMediaDataHandler (
    Media          theMedia,
    short          index,
    DataHandlerComponent  dataHandler );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

index

Identifies the data reference for this data handler. You provide the index value that corresponds to the data reference. You must set this parameter to 1.

dataHandler

The data handler for the media. This identifier is a component instance that specifies a connection to a data handler component, such as that returned by `GetMediaDataHandler` (I–425). If the data handler you specify cannot work with the data stored in the media, the function does not change the media’s data handler.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Your application should normally not call this function. The Movie Toolbox assigns a data handler to each media when you load a movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Selecting Media Handlers” (V–2754)

Related Java Methods

`quicktime.std.movies.media.Media.setDataHandler()`

See Also

For the corresponding get function, see `GetMediaDataHandler` (I–425).

SetMediaDataRef

Changes the file that the specified media identifies as the location for its data storage.

```
OSErr SetMediaDataRef (
    Media      theMedia,
    short      index,
    Handle      dataRef,
    OSType      dataRefType );
```

theMedia

Specifies the media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

index

A pointer to a short integer. The Movie Toolbox returns the index value that is assigned to the new data reference. Your application can use this index to identify the reference to other Movie Toolbox functions, such as `GetMediaDataRef` (I–427). As with all data reference functions, the index starts with 1. If the Movie Toolbox cannot add the data reference to the media, it sets the returned index value to 0.

dataRef

The data reference. This parameter contains a handle to the information that identifies the file that contains this media’s data. The type of information stored in that handle depends upon the value of the `dataRefType` parameter.

SetMediaDataRefAttributes

dataRefType

The type of data reference. If the data reference is an **alias**, you must set this parameter to `rAliasType`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Don’t call this function unless you have a really good reason. However, if you want to resolve your own missing data references, or you are developing a special-purpose kind of application, this function can be quite useful.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.media.Media.setDataRef()`

See Also

For the corresponding get function, see `GetMediaDataRef` (I–427).

SetMediaDataRefAttributes

Sets a data reference’s attributes.

```
OSErr SetMediaDataRefAttributes (
    Media      theMedia,
    short      index,
    long       dataRefAttributes );
```

theMedia

Specifies the media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

index

The index value that corresponds to the data reference. It must be less than or equal to the value that is returned by `GetMediaDataRefCount` (I–428).

dataRefAttributes

A flag (see below) that determines whether or not the data reference is the movie default.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

dataRefAttributes Constant

`kMovieAnchorDataRefIsDefault`

The data reference is the movie default data reference.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.media.Media.setDataRefAttributes()`

SetMediaDefaultDataRefIndex

Specifies which of a media’s data references is to be accessed during an editing session.

```
OSErr SetMediaDefaultDataRefIndex (
    Media    theMedia,
    short    index );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II-1120) and `GetTrackMedia` (I-535).

index

The data reference to access. Values of the `index` parameter range from 1 to the number of data references in the media. You can determine the number of data references by calling `GetMediaDataRefCount` (I-428). Once set, the default data reference index persists. Set this parameter to 0 to revert to the media’s default data reference.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

This function allows you to specify the index of the data reference to be edited. After calling this function, you can start editing that data reference by calling `BeginMediaEdits` (I-56).

SetMediaHandler

Version Notes

Before QuickTime 2.0, the Movie Toolbox did not allow the creation of tracks that had data in several files. Therefore, there was no mechanism for controlling which data reference was affected by a media editing session.

Programming Info

C interface file: `Movies.h`

Programming summary: “Adding Samples to Media Structures” (V–2752)

Related Java Methods

`quicktime.std.movies.media.Media.setDefaultDataRefIndex()`

SetMediaHandler

Assigns a specific media handler to a track.

```
OSErr SetMediaHandler (
    Media                theMedia,
    MediaHandlerComponent mH );
```

theMedia

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

mH

A reference to a media handler component. You obtain this reference from `GetMediaHandler`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Your application should not need to call this function. The Movie Toolbox assigns a media handler to each track when you load a movie.

Special Considerations

The Movie Toolbox closes the track’s previous media handler and then opens the new one. It is your responsibility to ensure that the media handler you specify can handle the data in the track.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Selecting Media Handlers” (V–2754)

Related Java Methods

`quicktime.std.movies.media.Media.setHandler()`

See Also

For the corresponding get function, see `GetMediaHandler` (I–432).

SetMediaInputMap

Replaces the media’s existing input map with a given input map.

```
OSErr SetMediaInputMap (
    Media          theMedia,
    QAtomContainer inputMap );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

inputMap

The media input map for this operation. If the input map is set to `NIL`, the media’s input map is reset to an empty input map.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Use this function to specify the media you want to set so you can modify or empty its input map. It makes a copy of the input map passed to it. The following sample code illustrates how to update an input map, using this function and `GetMediaInputMap` (I–435):

```
// SetMediaInputMap coding example
QAtomContainer inputMap;
QAtom inputAtom;
OSType inputType;

Media aVideoMedia = GetTrackMedia(aVideoTrack);
GetMediaInputMap (aVideoMedia, &inputMap);
```

SetMediaLanguage

```
QTInsertChild(inputMap, kParentAtomIsContainer, kTrackModifierInput,
              addedIndex, 0, 0, nil, &inputAtom);

inputType = kTrackModifierTypeClip;
QTInsertChild (inputMap, inputAtom, kTrackModifierType, 1, 0,
              sizeof(inputType), &inputType, nil);

SetMediaInputMap(aVideoMedia, inputMap);
QTDisposeAtomContainer(inputMap);
```

Special Considerations

Use `QTNewAtomContainer` (II–1203) to create an empty input map.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Manipulating Media Input Maps” (V–2753)

Related Java Methods

`quicktime.std.movies.media.Media.setInputMap()`

See Also

For the corresponding get function, see `GetMediaInputMap` (I–435).

SetMediaLanguage

Sets a media’s localized language or **region code**.

```
void SetMediaLanguage (
    Media    theMedia,
    short    language );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

language

The media’s language or **region code**.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

You should call this function only when you are creating a new media.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Alternate Tracks” (V–2738)

Related Java Methods

`quicktime.std.movies.media.Media.setLanguage()`

See Also

For the corresponding get function, see `GetMediaLanguage` (I–436).

SetMediaPlayHints

Provides information to the Movie Toolbox that can influence playback of a single media.

```
void SetMediaPlayHints (
    Media    theMedia,
    long     flags,
    long     flagsMask );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

flags

The optimizations that can be used with this media. Each bit in this parameter corresponds to a specific optimization; be sure to set unused flags to 0.

flagsMask

Indicates which flags in the `flags` parameter are to be considered in this operation. For each bit in the `flags` parameter that you want the Movie Toolbox to consider, you must set the corresponding bit in the `flagsMask` parameter to 1. Set unused flags to 0. This allows you to work with a single optimization without altering the settings of other flags.

function result

You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

SetMediaPlayHints

flags Constants

hintsScrubMode

Indicates that the Movie Toolbox can prefer to display **key frames** when the movie that uses this media is repositioned. This optimization is used only when a movie's rate is set to 0. If you set this flag to 1, the Movie Toolbox is free to display the nearest key frame when you set the movie's current time; the Movie Toolbox then moves to the appropriate frame as time permits. If you set this flag to 0, the Movie Toolbox displays the frame that corresponds to the new current time, even if that frame is not a key frame. By displaying key frames first, the Movie Toolbox can display data from temporally compressed movies much more quickly in response to changes to the movie's current time. This, in turn, can improve the liveliness of a movie control. For example, if the user is positioning in a stopped movie, the Movie Toolbox can display a key frame that corresponds to the new position without having to build up the image offscreen. In this manner, the user gets quicker feedback from your application.

hintsUseSoundInterp

Turns on sound interpolation; that is, tells the Sound Manager to use sound interpolation when playing back sound. In certain situations, this improves the sound quality to 11 kHz.

hintsAllowInterlace

Tells the Image Compression Manager to use the **interlace** option for image compressor and decompressor components.

hintsAllowBlacklining

Instructs the Movie Toolbox to use **blacklining** (displaying every other line of the image) when playing a movie. Blacklining increases the speed of movie playback while decreasing the image quality. This hint is ignored if the decompressor for the movie data does not support blacklining.

hintsDontPurge

Instructs the toolbox not to dispose of movie data after playing it. The toolbox leaves the data in memory, in a purgeable handle. This can enhance the playback of small movies that are looping. However, it may consume large amounts of memory and therefore affect the performance of the Memory Manager. Use this hint carefully.

hintsInactive

Tells the toolbox that the movie is not in an active window. This can allow the toolbox to allocate scarce system **resources** more efficiently. The movie controller component automatically sets this hint for all movies it manages.

`hintsHighQuality`

Specifies that the given movie or media should render the highest quality. For example, the video media handler should turn off fast **dithering** and use high-quality dithering. Because rendering at the highest quality takes much more time and memory than rendering at a lesser quality, this mode is usually not appropriate for real-time playback. However, since this mode generates higher quality images, it is useful when recompressing.

Discussion

This function accepts a flag in which you specify optimizations that the Movie Toolbox can use during movie playback. These optimizations apply to only the specified media.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V-2722)

Related Java Methods

`quicktime.std.movies.media.Media.setPlayHints()`

See Also

For the corresponding get function, see `GetMediaPlayHints` (I-439).

SetMediaPreferredChunkSize

Specifies a maximum chunk size for a media.

```
OSErr SetMediaPreferredChunkSize (
    Media    theMedia,
    long      maxChunkSize );
```

`theMedia`

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II-1120) and `GetTrackMedia` (I-535).

`maxChunkSize`

The maximum chunk size, in bytes.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

SetMediaPropertyAtom

Discussion

The term “chunk” refers to the collection of sample data that is added to a movie when you call `AddMediaSample` (I–27). When QuickTime loads a movie for playback, it loads the data a chunk at a time. Consequently, both the size and number of chunks in a movie can affect playback performance. The toolbox tries to optimize playback performance by consolidating adjacent sample references into a single chunk, up to the limit you prescribe with this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Adding Samples to Media Structures” (V–2752)

Related Java Methods

`quicktime.std.movies.media.Media.setPreferredChunkSize()`

See Also

For the corresponding get function, see `GetMediaPreferredChunkSize` (I–440).

SetMediaPropertyAtom

Sets the property atom container of a media handler.

```
OSErr SetMediaPropertyAtom (
    Media          theMedia,
    QTAtomContainer propertyAtom );
```

theMedia

A reference to the media handler for this operation.

propertyAtom

Specifies a QT atom container that contains the property atoms for the track associated with the media handler.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You can call this function to set properties for the track associated with the specified media handler. The contents of the QT atom container are defined by the media handler. Here is some sample code that uses this function to define the background color for a **sprite** track:

```
// SetMediaPropertyAtom coding example
// See “Discovering QuickTime,” page 360
if (bWithBackgroundPicture) {
    QTAtomContainer      qtacTrackProperties;
    RGBColor             rgbcBackColor;

    rgbcBackColor.red = EndianU16_NtoB(0x8000);
    rgbcBackColor.green = EndianU16_NtoB(0);
    rgbcBackColor.blue = EndianU16_NtoB(0xffff);

    // create a new atom container for sprite track properties
    QTNewAtomContainer(&qtacTrackProperties);

    // add an atom for the background color property
    QTInsertChild(qtacTrackProperties, 0,
        kSpriteTrackPropertyBackgroundColor, 1, 1, sizeof(RGBColor),
        &rgbcBackColor, NIL);

    // set the sprite track’s properties
    nErr = SetMediaPropertyAtom(media, qtacTrackProperties);

    QTDisposeAtomContainer(qtacTrackProperties);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Handler Properties” (V–2755)

Related Java Methods

`quicktime.std.movies.media.Media.setPropertyAtom()`

See Also

For the corresponding get function, see `GetMediaPropertyAtom` (I–440).

SetMediaQuality

Sets a media’s quality level value.

```
void SetMediaQuality (
    Media    theMedia,
```

SetMediaSampleDescription

```
short    quality );
```

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

quality

The media’s quality value. The quality value indicates the pixel depths at which the media can be played. This even applies to sound media. The low-order 6 bits of the quality value correspond to specific pixel depths. If a bit is set to 1, the media can be played at the corresponding depth. More than one of these bits may be set to 1. The Movie Toolbox uses this quality value to determine which track it selects to play on a given Macintosh computer. You should set this value only when you are creating a new media.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Alternate Tracks” (V–2738)

Related Java Methods

`quicktime.std.movies.media.Media.setQuality()`

See Also

For the corresponding get function, see `GetMediaQuality` (I–441).

SetMediaSampleDescription

Changes the contents of a particular `SampleDescription` (IV–2414) structure of a specified media.

```
OSErr SetMediaSampleDescription (
    Media                theMedia,
    long                 index,
    SampleDescriptionHandle descH );
```

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

index

The index of the `SampleDescription` (IV–2414) structure to be changed. This index corresponds to the `SampleDescription` structure itself, not the samples in the media. This long integer must be between 1 and the largest `SampleDescription` index.

descH

The handle to the `SampleDescription` (IV–2414) structure. If there is no description for the specified index, the function returns this handle unchanged.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Samples” (V–2742)

Related Java Methods

`quicktime.std.movies.media.Media.setSampleDescription()`

See Also

For the corresponding get function, see `GetMediaSampleDescription` (I–445).

SetMediaShadowSync

Creates an association between the indicated frame difference sample and a specified self-contained sample in a given media.

```
OSErr SetMediaShadowSync (
    Media    theMedia,
    long     frameDiffSampleNum,
    long     syncSampleNum );
```

theMedia

The media in which the **shadow sync** is to be created.

frameDiffSampleNum

Specifies a frame difference sample. The sample number is obtained from `MediaTimeToSampleNum` (II–913).

SetMediaTimeScale

syncSampleNum

Specifies a shadow **sync sample**. The sample number is obtained from `MediaTimeToSampleNum` (II–913).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V–2722)

Related Java Methods

`quicktime.std.movies.media.Media.setShadowSync()`

See Also

For the corresponding get function, see `GetMediaShadowSync` (I–453).

SetMediaTimeScale

Sets a media’s time scale.

```
void SetMediaTimeScale (  
    Media      theMedia,  
    TimeScale  timeScale );
```

`theMedia`

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

`timeScale`

The media’s new time scale.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Media Time” (V–2735)

Related Java Methods

`quicktime.std.movies.media.Media.setTimeScale()`

See Also

For the corresponding get function, see `GetMediaTimeScale` (I–455).

SetMovieActive

Activates or deactivates a movie.

```
void SetMovieActive (
    Movie      theMovie,
    Boolean    active );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

active

Activates or deactivates the movie. Set this parameter to `TRUE` to activate the movie; set this parameter to `FALSE` to deactivate the movie.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Disabling Movies and Tracks” (V–2724)

Related Java Methods

`quicktime.std.movies.Movie.setActive()`

See Also

For the corresponding get function, see `GetMovieActive` (I–456).

SetMovieActiveSegment

Defines a movie’s active segment.

```
void SetMovieActiveSegment (
```

SetMovieAnchorDataRef

```
Movie      theMovie,  
TimeValue  startTime,  
TimeValue  duration );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

startTime

A time value specifying the starting point of the active segment. Set this parameter to `-1` to make the entire movie active. In this case, the `SetMovieActiveSegment` function ignores the `duration` parameter.

duration

A time value that specifies the duration of the active segment. If you are making the entire movie active (by setting the `startTime` parameter to `-1`), the Movie Toolbox ignores this parameter.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

Your application defines the active segment by specifying the starting time and duration of the segment. These values must be expressed in the movie’s time coordinate system. By default, the entire movie is active.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V–2722)

Related Java Methods

`quicktime.std.movies.Movie.setActiveSegment()`

See Also

Your application can retrieve the information that defines a movie’s active segment by calling `GetMovieActiveSegment` (I–457).

SetMovieAnchorDataRef

Sets a movie’s anchor data reference and type.


```
OSErr SetMovieAnchorDataRef (
    Movie      theMovie,
    Handle     dataRef,
    OSType     dataRefType );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

dataRef

A handle to the data reference. The type of information to be placed in the handle depends upon the data reference type specified by `dataRefType`.

dataRefType

The type of data reference; see “Data References” (IV–2661).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

See Also

For the corresponding get function, see `GetMovieAnchorDataRef` (I–458).

SetMovieBox

Sets a movie’s boundary rectangle.

```
void SetMovieBox (
    Movie      theMovie,
    const Rect *boxRect );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

boxRect

A pointer to a rectangle that contains the coordinates of the new boundary rectangle.

SetMovieClipRgn

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

The Movie Toolbox changes the rectangle by modifying the translation and scale values of the movie’s matrix to accommodate the new boundary rectangle.

The movie box might not have its upper-left corner set at (0,0) in its display window when the movie is first loaded. Consequently, your application may need to adjust the position of the movie box so that it appears in the appropriate location within your application’s document window. If you don’t reset the movie position, the movie might not be visible when it starts playing. The following sample code demonstrates how to do this:

```
//Zeroing the boundary rectangle with SetMovieBox
GetMovieBox (movie, &movieBox);
OffsetRect (&movieBox, -movieBox.left, -movieBox.top);
SetMovieBox (movie, &movieBox);
```

Special Considerations

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

```
quicktime.std.movies.Movie.setBounds(),
quicktime.std.movies.Movie.setBox()
```

See Also

For the corresponding get function, see `GetMovieBox` (I–460).

SetMovieClipRgn

Establishes a movie’s clipping region.

```
void SetMovieClipRgn (
    Movie          theMovie,
    RgnHandle      theClip );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

theClip

A handle to the movie’s clipping region. The Movie Toolbox makes a copy of this region. Your application must dispose of the region referred to by this parameter when you are done with it. Set this parameter to `NIL` to disable clipping for the movie.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

The clipping region is saved with the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Movie.setClipRgn()`

See Also

For the corresponding get function, see `GetMovieClipRgn` (I–462).

SetMovieColorTable

Associates a `ColorTable` (IV–2210) structure with a movie.

```
OSErr SetMovieColorTable (
    Movie      theMovie,
    CTabHandle ctab );
```

theMovie

The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

SetMovieCoverProcs

`ctab`

A handle to the `ColorTable` (IV–2210) structure. Set this parameter to `NIL` to remove the movie’s `ColorTable` structure.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The `ColorTable` (IV–2210) structure you supply may be used to modify the palette of indexed display devices at playback time. If you are using the movie controller, be sure to set the `mcFlagsUseWindowPalette` flag. If you are not using the movie controller, you should retrieve the movie’s `ColorTable` structure, using `GetMovieColorTable` (I–463), and supply it to the Palette Manager.

Special Considerations

The toolbox makes a copy of the `ColorTable` structure, so it is your responsibility to dispose of the structure when you are done with it. If the movie already has a color table, the toolbox uses the new table to replace the old one.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Movie.setColorTable()`

See Also

For the corresponding get function, see `GetMovieColorTable` (I–463).

`CopyMovieSettings` (I–140) copies the movie’s color table, along with other settings information.

SetMovieCoverProcs

Sets the callbacks invoked when a movie is covered or uncovered.

```
void SetMovieCoverProcs (
    Movie          theMovie,
    MovieRgnCoverUPP uncoverProc,
    MovieRgnCoverUPP coverProc,
    long           refcon );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

uncoverProc

Points to a `MovieRgnCoverProc` (III–2108) callback. This function is called whenever one of your movie’s tracks is removed from the screen or resized, revealing a previously hidden screen region. If you want to remove this uncover function, set this parameter to `NIL`. When the `uncoverProc` parameter is `NIL` the function uses the default uncover function, which erases the uncovered area.

coverProc

Points to a `MovieRgnCoverProc` (III–2108) callback. The Movie Toolbox calls this function whenever one of your movies covers a portion of the screen. If you want to remove the cover function, set this parameter to `NIL`. When the `uncoverProc` parameter is `NIL` the function uses the default cover function, which does nothing.

refcon

Specifies a reference constant. Use this parameter to point to a data structure containing any information your callbacks need.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

If a movie with semi-transparent tracks has a movie uncover procedure, set with this function, the uncover procedure is called before each frame to fill or erase the background.

Version Notes

Before QuickTime 1.6.1, the Movie Toolbox performed the erase, which limited a cover procedure-aware application’s options.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Progress and Cover Functions” (V–2726)

See Also

For the corresponding get function, see `GetMovieCoverProcs` (I–464).

SetMovieDefaultDataRef

SetMovieDefaultDataRef

Sets a movie's default data reference and type.

```
OSErr SetMovieDefaultDataRef (
    Movie      theMovie,
    Handle     dataRef,
    OSType     dataRefType );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

dataRef

A handle to the data reference. The type of information to be placed in the handle depends upon the data reference type specified by *dataRefType*.

dataRefType

The type of data reference; see “Data References” (IV–2661).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.Movie.setDefaultDataRef()`

See Also

For the corresponding get function, see `GetMovieDefaultDataRef` (I–467).

SetMovieDisplayClipRgn

Establishes a movie's current display clipping region.

```
void SetMovieDisplayClipRgn (
    Movie      theMovie,
    RgnHandle  theClip );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

theClip

A handle to the movie’s display clipping region as a `MacRegion` (IV–2303) structure. The Movie Toolbox makes a copy of this region. Your application must dispose of the region referred to by this parameter when you are done with it. Set this parameter to `NIL` to disable a movie’s clipping region.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

The display clipping region is not saved with the movie. You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Movie.setDisplayClipRgn()`

See Also

For the corresponding get function, see `GetMovieDisplayClipRgn` (I–469).

SetMovieDrawingCompleteProc

Assigns a drawing-complete function to a movie.

```
void SetMovieDrawingCompleteProc (
    Movie          theMovie,
    long           flags,
    MovieDrawingCompleteUPP proc,
    long           refCon );
```

theMovie

The movie for this operation.

SetMovieDrawingCompleteProc

flags

Contains flags (see below) that control when your drawing complete function is called.

proc

A pointer to your MovieDrawingCompleteProc (III–2100) callback. Set this parameter to NIL if you want to remove your callback.

refCon

The reference constant you supplied when your application called your callback. Use this parameter to point to a data structure containing any information your callback needs.

function result You can access error returns from this function through GetMoviesError (I–492) and GetMoviesStickyError (I–494). See “Error Codes” (IV–2663).

flags Constants

movieDrawingCallWhenChanged

Specifies that the toolbox should call your drawing-complete function only when the movie has changed.

movieDrawingCallAlways

Specifies that the toolbox should call your drawing-complete function every time your application calls MoviesTask (II–973).

Discussion

The Movie Toolbox calls this function based upon guidelines you establish when you assign the function to the movie.

Special Considerations

Some media handlers may take less efficient playback paths when a drawing-complete function is used, so it should be used only when absolutely necessary.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Working With Progress and Cover Functions” (V–2726)

Related Java Methods

quicktime.std.movies.Movie.setDrawingCompleteProc(),

quicktime.std.movies.Movie.removeDrawingCompleteProc()

SetMovieGWorld

Establishes a movie's display coordinate system by setting the graphics world for displaying the movie.

```
void SetMovieGWorld (
    Movie      theMovie,
    CGrafPtr   port,
    GDHandle    gdh );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

port

Points to the movie's `CGrafPort` (IV–2168) structure or graphics world. Set this parameter to `NIL` to use the current graphics port.

gdh

A handle to the movie's `GDevice` (IV–2264) structure. Set this parameter to `NIL` to use the current device. If the port parameter specifies a graphics world, set this parameter to `NIL` to use that graphics world's graphics device.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Special Considerations

When you use this function, the Movie Toolbox remembers the current background color and background pattern. These are used for erasing in the default movie uncover function; see `SetMovieCoverProcs` (III–1614).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Movie.setGWorld()`

See Also

For the corresponding get function, see `GetMovieGWorld` (I–471).

SetMovieLanguage

SetMovieLanguage

Specifies a movie's localized language or **region code**.

```
void SetMovieLanguage (
    Movie    theMovie,
    long     language );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

language

The movie's language or **region code**; see “Localization Codes” (IV–2673).

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

The Movie Toolbox examines the movie's alternate groups and selects and enables appropriate tracks. If the Movie Toolbox cannot find an appropriate track, it does not change the movie's language.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Alternate Tracks” (V–2738)

Related Java Methods

`quicktime.std.movies.Movie.setLanguage()`

SetMovieMasterClock

Assigns a clock component to a movie.

```
void SetMovieMasterClock (
    Movie          theMovie,
    Component      clockMeister,
    const TimeRecord *slaveZero );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

`clockMeister`

The clock component to be assigned to this movie. Your application can obtain this component identifier from `FindNextComponent` (I–360).

`slaveZero`

A pointer to the time, in the clock’s time scale, that corresponds to a 0 time value for the movie. This parameter allows you to set an offset between the clock component and the time base of the movie. Set this parameter to `NIL` if there is no offset.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

Don’t use `SetTimeBaseMasterClock` (III–1649) to assign a clock component to a movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Disposing of Time Bases” (V–2759)

Related Java Methods

`quicktime.std.movies.Movie.setMasterClock()`

SetMovieMasterTimeBase

Assigns a master time base to a movie.

```
void SetMovieMasterTimeBase (
    Movie          theMovie,
    TimeBase       tb,
    const TimeRecord *slaveZero );
```

SetMovieMatrix

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

tb

The master time base to be assigned to this movie. Your application obtains this time base identifier from `NewTimeBase` (II–1119).

slaveZero

A pointer to the time, in the time scale of the master time base, that corresponds to a 0 time value for the movie. This parameter allows you to set an offset between the movie and the master time base. Set this parameter to `NIL` if there is no offset.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Disposing of Time Bases” (V–2759)

Related Java Methods

`quicktime.std.movies.Movie.setMasterTimeBase()`

SetMovieMatrix

Sets a movie’s transformation matrix.

```
void SetMovieMatrix (
    Movie          theMovie,
    const MatrixRecord *matrix );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

matrix

A pointer to the `MatrixRecord` (IV–2304) structure for the movie. If you set this parameter to `NIL`, the Movie Toolbox uses the identity matrix.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

The Movie Toolbox uses a movie’s matrix to map a movie from its display coordinate system to its graphics world.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Movie.setMatrix()`

See Also

For the corresponding get function, see `GetMovieMatrix` (I–478).

SetMoviePlayHints

Provides information to the Movie Toolbox that can influence movie playback.

```
void SetMoviePlayHints (
    Movie    theMovie,
    long     flags,
    long     flagsMask );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

flags

The optimizations that can be used with this movie. Each bit in the `flags` parameter corresponds to a specific optimization (see below). Be sure to set unused flags to 0.

flagsMask

Indicates which flags in the `flags` parameter are to be considered in this operation. For each bit in the `flags` parameter that you want the Movie Toolbox to consider, you must set the corresponding bit in the `flagsMask` parameter to 1.

SetMoviePlayHints

Set unused flags to 0. This allows you to work with a single optimization without altering the settings of other flags.

function result You can access error returns from this function through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494). See “Error Codes” (IV-2663).

flags Constants

`hintsScrubMode`

Indicates that the Movie Toolbox can prefer to display **key frames** when the movie that uses this media is repositioned. This optimization is used only when a movie’s rate is set to 0. If you set this flag to 1, the Movie Toolbox is free to display the nearest key frame when you set the movie’s current time; the Movie Toolbox then moves to the appropriate frame as time permits. If you set this flag to 0, the Movie Toolbox displays the frame that corresponds to the new current time, even if that frame is not a key frame. By displaying key frames first, the Movie Toolbox can display data from temporally compressed movies much more quickly in response to changes to the movie’s current time. This, in turn, can improve the liveliness of a movie control. For example, if the user is positioning in a stopped movie, the Movie Toolbox can display a key frame that corresponds to the new position without having to build up the image offscreen. In this manner, the user gets quicker feedback from your application.

`hintsUseSoundInterp`

Turns on sound interpolation; that is, tells the Sound Manager to use sound interpolation when playing back sound. In certain situations, this improves the sound quality to 11 kHz.

`hintsAllowInterlace`

Tells the Image Compression Manager to use the **interlace** option for image compressor and decompressor components.

`hintsAllowBlacklining`

Instructs the Movie Toolbox to use **blacklining** (displaying every other line of the image) when playing a movie. Blacklining increases the speed of movie playback while decreasing the image quality. This hint is ignored if the decompressor for the movie data does not support blacklining.

`hintsDontPurge`

Instructs the toolbox not to dispose of movie data after playing it. The toolbox leaves the data in memory, in a purgeable handle. This can enhance the playback of small movies that are looping. However, it may consume large amounts of memory and therefore affect the performance of the Memory Manager. Use this hint carefully.

hintsInactive

Tells the toolbox that the movie is not in an active window. This can allow the toolbox to allocate scarce system **resources** more efficiently. The movie controller component automatically sets this hint for all movies it manages.

hintsHighQuality

Specifies that the given movie or media should render the highest quality. For example, the video media handler should turn off fast **dithering** and use high-quality dithering. Because rendering at the highest quality takes much more time and memory than rendering at a lesser quality, this mode is usually not appropriate for real-time playback. However, since this mode generates higher quality images, it is useful when recompressing.

Discussion

This function accepts a flag in which you specify optimizations that the Movie Toolbox can use during movie playback. These optimizations apply to all of the media structures used by the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V-2722)

Related Java Methods

`quicktime.std.movies.Movie.setPlayHints()`

SetMoviePosterTime

Sets the **poster** time for the movie.

```
void SetMoviePosterTime (
    Movie      theMovie,
    TimeValue  posterTime );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), and `NewMovieFromHandle` (II-1084).

posterTime

The starting time for the movie frame that contains the **poster** image, expressed in the movie’s time coordinate system.

SetMoviePreferredRate

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

Since a movie **poster** is a still frame, it is defined by a point in time within the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V–2718)

Related Java Methods

`quicktime.std.movies.Movie.setPosterTime()`

See Also

Your application can retrieve a **poster**’s time by calling `GetMoviePosterTime` (I–485).

SetMoviePreferredRate

Specifies a movie’s default playback rate.

```
void SetMoviePreferredRate (  
    Movie    theMovie,  
    Fixed    rate );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

rate

The new movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Preferred Movie Settings” (V–2721)

Related Java Methods

`quicktime.std.movies.Movie.setPreferredRate()`

See Also

You can set the current playback rate of a movie by calling `SetMovieRate` (III–1631).

SetMoviePreferredVolume

Sets a movie’s preferred volume setting.

```
void SetMoviePreferredVolume (
    Movie    theMovie,
    short    volume );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

volume

The preferred volume setting of the movie. The volume parameter must contain a 16-bit, fixed-point number that contains the movie’s default volume. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from –1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. You may find the constants shown below useful.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

volume Constants

`kFullVolume`

Sets the track to full volume (constant value is 1.0).

`kNoVolume`

Sets the track to no volume (constant value is 0.0).

SetMoviePreviewMode

Discussion

A movie's tracks may have their own volume settings. Use `SetTrackVolume` (III–1669) to set the volume of an individual track. A track's volume is scaled by the movie's volume to produce the track's final volume.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Preferred Movie Settings” (V–2721)

Related Java Methods

`quicktime.std.movies.Movie.setPreferredVolume()`

See Also

For the corresponding get function, see `GetMoviePreferredVolume` (I–486).

SetMoviePreviewMode

Places a movie into and out of **preview** mode.

```
void SetMoviePreviewMode (
    Movie      theMovie,
    Boolean    usePreview );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

usePreview

The movie's mode. Set this parameter to `TRUE` to place the movie into **preview** mode. Set this parameter to `FALSE` to place the movie into normal playback mode.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

When a movie is in **preview** mode, only those tracks identified as preview tracks are serviced. You specify how a track is used by calling `SetTrackUsage` (III–1668).

When you place a movie into preview mode, the Movie Toolbox sets the active

movie segment to be the preview segment of the movie. When you take a movie out of preview mode and place it back in normal playback mode, the toolbox sets the active movie segment to be the entire movie. For information about working with active movie segments, see `PrerollMovie` (II–1142).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V–2718)

Related Java Methods

`quicktime.std.movies.Movie.setPreviewMode()`

See Also

For the corresponding get function, see `GetMoviePreviewMode` (I–487).

SetMoviePreviewTime

Defines the starting time and duration of the movie’s **preview**.

```
void SetMoviePreviewTime (
    Movie          theMovie,
    TimeValue      previewTime,
    TimeValue      previewDuration );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

previewTime

A time value that specifies the **preview**’s starting time.

previewDuration

A time value that specifies the **preview**’s duration.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

SetMovieProgressProc

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V–2718)

Related Java Methods

`quicktime.std.movies.Movie.setPreviewTime()`

See Also

For the corresponding get function, see `GetMoviePreviewTime` (I–487).

SetMovieProgressProc

Attaches a **progress function** to a movie.

```
void SetMovieProgressProc (
    Movie          theMovie,
    MovieProgressUPP p,
    long           refcon );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

p

Points to your `MovieProgressProc` (III–2105) callback. To remove a movie’s **progress function**, set this parameter to `NIL`. Set this parameter to `–1` for the Movie Toolbox to provide a default progress function.

refcon

Specifies a reference constant. Use this parameter to point to a data structure containing any information your callback needs.

function result

You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

The Movie Toolbox calls your function only during long operations. It ensures that your **progress function** is called regularly, but not too often.

The following Movie Toolbox functions use progress functions:

`ConvertFileToMovieFile` (I–129), `CutMovieSelection` (I–166), `CopyMovieSelection` (I–139), `AddMovieSelection` (I–38), and `InsertMovieSegment` (II–762).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Progress and Cover Functions” (V–2726)

Related Java Methods

`quicktime.std.movies.Movie.setProgressProc()`

See Also

For the corresponding get function, see `GetMovieProgressProc` (I–488).

S

SetMoviePropertyAtom

Sets a movie's property atom.

```
OSErr SetMoviePropertyAtom (
    Movie          theMovie,
    QTAtomContainer propertyAtom );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

propertyAtom

A property atom.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Movies.h`

See Also

For the corresponding get function, see `GetMoviePropertyAtom` (I–489).

SetMovieRate

Sets a movie's playback rate.

SetMovieSelection

```
void SetMovieRate (
    Movie      theMovie,
    Fixed      rate );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

rate

The new movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Time” (V–2732)

Related Java Methods

`quicktime.std.movies.Movie.setRate()`

See Also

For the corresponding get function, see `GetMovieRate` (I–490).

SetMovieSelection

Sets a movie’s current selection.

```
void SetMovieSelection (
    Movie      theMovie,
    TimeValue  selectionTime,
    TimeValue  selectionDuration );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

selectionTime

A time value specifying the starting point of the current selection.

selectionDuration

A time value that specifies the duration of the current selection.

function result You can access error returns from this function through GetMoviesError (I–492) and GetMoviesStickyError (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “High-Level Movie Editing Functions” (V–2746)

Related Java Methods

quicktime.std.movies.Movie.setSelection()

See Also

For the corresponding get function, see GetMovieSelection (I–491).

SetMoviesErrorProc

Performs custom error notification.

```
void SetMoviesErrorProc (
    MoviesErrorUPP    errProc,
    long              refcon );
```

errProc

A MoviesErrorProc (III–2109) callback.

refcon

A reference constant value. The Movie Toolbox passes this reference constant to your MoviesErrorProc (III–2109) callback each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

function result You can access error returns from this function through GetMoviesError (I–492) and GetMoviesStickyError (I–494). See “Error Codes” (IV–2663).

Discussion

Your application must identify its custom error-notification function to the Movie Toolbox. Once you have identified an error-notification function, the Movie Toolbox calls your function each time the current error value is to be set to a nonzero

SetMovieTime

value. The Movie Toolbox calls your error-notification function only in response to errors generated by the Movie Toolbox.

Special Considerations

Error-notification functions can be especially useful when you are debugging your program. The Movie Toolbox manages the **sticky error** value.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Error Functions” (V–2712)

SetMovieTime

Changes a movie’s current time.

```
void SetMovieTime (
    Movie          theMovie,
    const TimeRecord *newtime );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

newtime

A pointer to a `TimeRecord` (IV–2486) structure containing the new time.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

The Movie Toolbox saves the movie’s current time when you save the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Time” (V–2732)

Related Java Methods

`quicktime.std.movies.Movie.setTime()`

See Also

For the corresponding get function, see `GetMovieTime` (I–495).

SetMovieTimeScale

Establishes a movie’s time scale.

```
void SetMovieTimeScale (
    Movie      theMovie,
    TimeScale  timeScale );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

timeScale

The movie’s new time scale.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Time” (V–2732)

Related Java Methods

`quicktime.std.movies.Movie.setTimeScale()`

See Also

For the corresponding get function, see `GetMovieTimeScale` (I–496).

SetMovieTimeValue

Sets a movie’s time value.

```
void SetMovieTimeValue (
    Movie      theMovie,
    TimeValue  newtime );
```

SetMovieVideoOutput

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

newtime

The new time value. You must ensure that the time value is in the movie’s time scale.

function result

You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Time” (V–2732)

Related Java Methods

`quicktime.std.movies.Movie.setTimeValue()`

SetMovieVideoOutput

Indicates to the ICM the video output component being used with a given movie.

```
void SetMovieVideoOutput (
    Movie          theMovie,
    ComponentInstance vout );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

vout

The video output component. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131). Call the function and pass `NIL` in this parameter as soon as the video output component is no longer in use.

Discussion

As soon as you turn on the echo port on any video output component, you should make this call so the ICM keeps track of the video output in use.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `Movies.h`

SetMovieVolume

Sets a movie's current volume.

```
void SetMovieVolume (
    Movie    theMovie,
    short    volume );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

volume

The current volume setting of the movie represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from –1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. You can use the constants shown below.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Volume Constants

`kFullVolume`

Sets the track to full volume (constant value is 1.0).

`kNoVolume`

Sets the track to no volume (constant value is 0.0).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Sound Volume” (V–2732)

Related Java Methods

`quicktime.std.movies.Movie.setVolume()`

SetPosterBox

See Also

For the corresponding get function, see `GetMovieVolume` (I–500).

SetPosterBox

Sets a **poster**'s boundary rectangle.

```
void SetPosterBox (  
    Movie          theMovie,  
    const Rect     *boxRect );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

boxRect

A pointer to a `Rect` (IV–2399) structure. The Movie Toolbox sets the **poster**'s boundary rectangle to the coordinates specified in the structure referred to by this parameter.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

You define the **poster**'s image by specifying a time in the movie, using `SetMoviePosterTime` (III–1625). You specify the size and position of the poster image with this function. If you don't specify a boundary rectangle for the poster, the Movie Toolbox uses the movie's matrix when it displays the poster.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V–2718)

Related Java Methods

`quicktime.std.movies.Movie.setPosterBox()`

See Also

For the corresponding get function, see `GetPosterBox` (I–505).

SetQuickTimePreference

Sets a particular preference in the QuickTime preferences.

```
OSErr SetQuickTimePreference (
    OSType          preferenceType,
    QTAtomContainer preferenceAtom );
```

preferenceType

The type of preference to set (see below); also see “Atom ID Codes” (IV–2649).

preferenceAtom

A QT atom containing the preference information.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

preferenceType Constants

ConnectionSpeedPrefsType

The connection speed currently set in the QuickTime Preferences; the atom type passed in preferenceAtom is 'cspd'.

BandwidthManagementPrefsType

The user’s current **band**width management preference; the atom type passed in preferenceAtom is 'bwmg'.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

See Also

For the corresponding get function, see GetQuickTimePreference (I–507).

SetSequenceProgressProc

Installs a progress procedure for a sequence.

```
OSErr SetSequenceProgressProc (
    ImageSequence      seqID,
    ICMPProgressProcRecord *progressProc );
```

SetSoundOutputInfo

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (I–119) or `DecompressSequenceBegin` (I–238).

progressProc

A pointer to an `ICMProgressProcRecord` (IV–2284) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function allows you to set an `ICMProgressProc` (III–2093) callback for a compression or decompression sequence, just as you can set a progress procedure when compressing or decompressing a still image.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Sequences” (V–2792)

SetSoundOutputInfo

Sets information about the current sound environment.

```
OSErr SetSoundOutputInfo (
    Component    outputDevice,
    OSType       selector,
    const void    *infoPtr );
```

outputDevice

The identifier of a sound output component. Your software obtains this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131). To access the default output device, pass `NIL`.

selector

The selector for the information you want to set; see “Sound Information Selectors” (IV–2693).

infoPtr

A pointer to the sound device information to be passed.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

See Also

For the corresponding get function, see GetSoundOutputInfo (I-511).

SetSoundPreference

Stores a block of preferences data in a sound **resource** file.

```
OSErr SetSoundPreference (
    OSType    theType,
    Str255    name,
    Handle     settings );
```

theType

The **resource** type of the preferences resource.

name

The **resource** name of the preferences resource.

settings

A handle to the data to be set in the preferences **resource**.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

See Also

For the corresponding get function, see GetSoundPreference (I-512).

SetSpriteProperty

Sets the specified property of a **sprite**.

```
OSErr SetSpriteProperty (
    Sprite    theSprite,
    long      propertyType,
    void      *propertyValue );
```

SetSpriteProperty

theSprite

The **sprite** for this operation.

propertyType

The property you want to modify (see below).

propertyValue

The new value of the property. Depending on the property type, you set the `propertyValue` parameter to either a pointer to the property value or the property value itself, cast as a void pointer.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

propertyType Constants

`kSpritePropertyMatrix`

The `propertyValue` parameter is the **sprite**’s `MatrixRecord` (IV–2304) structure.

`kSpritePropertyImageDescription`

The `propertyValue` parameter is an `ImageDescriptionHandle`, which is a handle to the **sprite**’s `ImageDescription` (IV–2285) structure.

`kSpritePropertyImageDataPtr`

The `propertyValue` parameter is a `Ptr`, which is a pointer to the **sprite**’s image data.

`kSpritePropertyVisible`

The `propertyValue` parameter is a `Boolean` that is `TRUE` if the **sprite** is visible, `FALSE` otherwise.

`kSpritePropertyLayer`

The `propertyValue` parameter is the **sprite**’s layer number.

`kSpritePropertyGraphicsMode`

The `propertyValue` parameter is the **sprite**’s `ModifierTrackGraphicsModeRecord` (IV–2311).

Discussion

You animate a **sprite** by modifying its properties, using this function. It invalidates the **sprite**’s **sprite world** as needed. Here is sample code that uses this function to modify a **sprite**’s properties:

```
// SetSpriteProperty coding example
// See “Discovering QuickTime,” page 345
#define kNumSprites          4
#define kNumSpaceShipImages 24
```



```

Rect          gBounceBox;
Sprite        gSprites[kNumSprites];
Rect          gDestRects[kNumSprites];
Point         gDeltas[kNumSprites];
short         gCurrentImages[kNumSprites];
Handle        gCompressedPictures[kNumSpaceShipImages];

void MyMoveSprites (void)
{
    short      nIndex;
    MatrixRecord matrix;

    SetIdentityMatrix(&matrix);

    // for each sprite
    for (nIndex = 0; nIndex < kNumSprites; nIndex++) {
        // modify the sprite's matrix
        OffsetRect(&gDestRects[nIndex], gDeltas[nIndex].h,
                  gDeltas[nIndex].v);

        if ((gDestRects[nIndex].right >= gBounceBox.right) ||
            (gDestRects[nIndex].left <= gBounceBox.left))
            gDeltas[nIndex].h = -gDeltas[nIndex].h;

        if ((gDestRects[nIndex].bottom >= gBounceBox.bottom) ||
            (gDestRects[nIndex].top <= gBounceBox.top))
            gDeltas[nIndex].v = -gDeltas[nIndex].v;

        matrix.matrix[2][0] = ((long)gDestRects[nIndex].left << 16);
        matrix.matrix[2][1] = ((long)gDestRects[nIndex].top << 16);

        SetSpriteProperty(gSprites[nIndex], kSpritePropertyMatrix,
                          &matrix);

        // change the sprite's image
        gCurrentImages[nIndex]++;
        if (gCurrentImages[nIndex] >= (kNumSpaceShipImages *
                                         (nIndex+1)))
            gCurrentImages[nIndex] = 0;
        SetSpriteProperty(gSprites[nIndex], kSpritePropertyImageDataPtr,
                          *gCompressedPictures[gCurrentImages[nIndex] / (nIndex+1)]);
    }
}

```

SetSpriteWorldClip

```
    }  
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Manipulating Sprites” (V–2901)

Related Java Methods

```
quicktime.std.anim.Sprite.setMatrix(),  
quicktime.std.anim.Sprite.setImageDescription(),  
quicktime.std.anim.Sprite.setImageData(),  
quicktime.std.anim.Sprite.setVisible(),  
quicktime.std.anim.Sprite.setLayer(),  
quicktime.std.anim.Sprite.setGraphicsMode(),  
quicktime.std.anim.Sprite.setImageDataSize()
```

See Also

For the corresponding get function, see `GetSpriteProperty` (I–512).

SetSpriteWorldClip

Sets a **sprite** world’s clip shape to the specified region.

```
OSErr SetSpriteWorldClip (  
    SpriteWorld    theSpriteWorld,  
    RgnHandle      clipRgn );
```

`theSpriteWorld`

The **sprite** world for this operation.

`clipRgn`

The new clip shape for the **sprite** world. The clip shape should be specified in the sprite world’s source space, the coordinate system of the sprite layer’s graphics world before the sprite world’s matrix is applied to it. You may pass a value of `NIL` for this parameter to indicate that there is no longer a clip shape for the sprite world. This means that the whole area is drawn.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You call this function to change the clip shape of a **sprite** world. The specified region is owned by the caller and is not copied by this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working with Sprite Worlds” (V–2900)

Related Java Methods

`quicktime.app.anim.SWCompositor.setClip()`,
`quicktime.std.anim.SpriteWorld.setClip()`

S

SetSpriteWorldFlags

Sets flags that govern the behavior of a **sprite** world.

```
OSErr SetSpriteWorldFlags (
    SpriteWorld    spriteWorld,
    long           flags,
    long           flagsMask );
```

spriteWorld

The **sprite** world for this operation.

flags

Constants (see below) that govern **sprite** world behavior.

flagsMask

Indicates which flags in the *flags* parameter are to be considered in this operation. For each bit in the *flags* parameter that you want the Movie Toolbox to consider, set the corresponding bit in the *flagsMask* parameter to 1. Set unused flags to 0. This allows you to work with a single optimization without altering the settings of other flags.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

flags Constants

`kScaleSpritesToScaleWorld`

Undocumented

SetSpriteWorldGraphicsMode

kSpriteWorldHighQuality

Undocumented

kSpriteWorldDontAutoInvalidate

Undocumented

kSpriteWorldInvisible

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Related Java Methods

quicktime.std.anim.SpriteWorld.setFlags()

SetSpriteWorldGraphicsMode

Sets the graphics transfer mode for a **sprite** world.

```
OSErr SetSpriteWorldGraphicsMode (
    SpriteWorld    theSpriteWorld,
    long           mode,
    const RGBColor *opColor );
```

theSpriteWorld

The **sprite** world for this operation.

mode

A long integer; see “Graphics Transfer Modes” (IV–2670).

opColor

A pointer to an RGBColor (IV–2403) structure. This is the blend value for blends and the transparent color for transparent operations. The toolbox supplies this value to QuickDraw when you draw in addPin, subPin, blend, transparent, or graphicsModeStraightAlphaBlend mode.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.anim.SpriteWorld.setGraphicsMode()`

SetSpriteWorldMatrix

Sets a **sprite** world's matrix to the specified matrix.

```
OSErr SetSpriteWorldMatrix (
    SpriteWorld    theSpriteWorld,
    const MatrixRecord *matrix );
```

theSpriteWorld

The **sprite** world for this operation.

matrix

A pointer to the new matrix for the **sprite** world. Transformations may include translation, scaling, rotation, skewing, and perspective. You may pass a value of `NIL` to set the sprite world's matrix to an identity matrix.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working with Sprite Worlds” (V-2900)

Related Java Methods

`quicktime.std.anim.SpriteWorld.setMatrix()`

SetSysBeepVolume

Sets the sound volume of the system beep.

```
OSErr SetSysBeepVolume (
    long    level );
```

SetTimeBaseFlags

level

The volume level to be set. The value may range from 0x0000 (silence) to 0x0100 (full volume).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding get function, see `GetSysBeepVolume` (I–514).

SetTimeBaseFlags

Sets the contents of the control flags of a time base.

```
void SetTimeBaseFlags (
    TimeBase    tb,
    long        timeBaseFlags );
```

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II–1119).

timeBaseFlags

The control flags for this time base (see below). You may set only one flag to 1. Be sure to set unused flags to 0.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

timeBaseFlags Constants

`loopTimeBase`

Indicates whether the time base loops. If you set this flag to 1 and the rate is positive, the time base loops back and restarts from its start time when it reaches its stop time. If you set this flag to 1 and the rate is negative, the time base loops to its stop time. If you set the flag to 0, the movie stops when it reaches the end.

palindromeLoopTimeBase

Indicates whether the time base loops in a palindrome fashion. Palindrome looping causes a time base to move alternately forward and backward. Set this flag to 1 to cause the time base to loop in this manner.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Time Base Values” (V–2760)

Related Java Methods

`quicktime.std.clocks.TimeBase.setFlags()`

See Also

For the corresponding get function, see `GetTimeBaseFlags` (I–515).

SetTimeBaseMasterClock

Assigns a clock component to a time base.

```
void SetTimeBaseMasterClock (
    TimeBase      slave,
    Component      clockMeister,
    const TimeRecord *slaveZero );
```

slave

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II–1119).

clockMeister

The clock component to be assigned to this time base. Your application can obtain this component identifier from `FindNextComponent` (I–360).

slaveZero

A pointer to the time, in the clock’s time scale, that corresponds to a 0 time value for the slave time base. This parameter allows you to set an offset between the time base and the clock component. Set this parameter to `NIL` if there is no offset.

function result

You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

SetTimeBaseMasterTimeBase

Discussion

A time base derives its time from either a clock component or from another time base. Don't use this function to assign a clock to a movie's time base.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Creating and Disposing of Time Bases" (V-2759)

Related Java Methods

`quicktime.std.clocks.TimeBase.setMasterClock()`

See Also

For the corresponding get function, see `GetTimeBaseMasterClock` (I-516).

SetTimeBaseMasterTimeBase

Assigns a master time base to a time base.

```
void SetTimeBaseMasterTimeBase (
    TimeBase      slave,
    TimeBase      master,
    const TimeRecord *slaveZero );
```

slave

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II-1119).

master

The master time base to be assigned to this time base. Your application obtains this time base identifier from `NewTimeBase` (II-1119).

slaveZero

A pointer to the time, in the time scale of the master time base, that corresponds to a 0 time value for the slave time scale. This parameter allows you to set an offset between the time base and the master time base. Set this parameter to `NIL` if there is no offset.

function result

You can access error returns from this function through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494). See "Error Codes" (IV-2663).

Discussion

A time base derives its time from either a clock component or another time base. Don't use this function to assign a master time base to a movie's time base.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Creating and Disposing of Time Bases" (V-2759)

Related Java Methods

`quicktime.std.clocks.TimeBase.setMasterTimeBase()`

See Also

For the corresponding get function, see `GetTimeBaseMasterTimeBase` (I-517).

SetTimeBaseRate

Sets the rate of a time base.

```
void SetTimeBaseRate (
    TimeBase  tb,
    Fixed     r );
```

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II-1119).

r

The rate of the time base.

function result You can access error returns from this function through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494). See "Error Codes" (IV-2663).

Discussion

Rates may be set to negative values. Negative rates cause time to move backward for the time base.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Working With Time Base Values" (V-2760)

SetTimeBaseStartTime

Related Java Methods

`quicktime.std.clocks.TimeBase.setRate()`

See Also

For the corresponding get function, see `GetTimeBaseRate` (I-518).

SetTimeBaseStartTime

Sets the start time of a time base.

```
void SetTimeBaseStartTime (
    TimeBase      tb,
    const TimeRecord *tr );
```

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II-1119).

tr

A pointer to a `TimeRecord` (IV-2486) structure that contains the start time value.

function result You can access error returns from this function through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494). See “Error Codes” (IV-2663).

Discussion

The start time defines the time base’s minimum time value. You must specify the new start time in a **time structure**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Time Base Values” (V-2760)

Related Java Methods

`quicktime.std.clocks.TimeBase.setStartTime()`

See Also

For the corresponding get function, see `GetTimeBaseStartTime` (I-518).

SetTimeBaseStopTime

Sets the stop time of a time base.

```
void SetTimeBaseStopTime (
    TimeBase      tb,
    const TimeRecord *tr );
```

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II–1119).

tr

A pointer to a `TimeRecord` (IV–2486) structure that contains the stop time value.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

The stop time defines the time base’s maximum time value. You must specify the new stop time in a **time structure**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Time Base Values” (V–2760)

Related Java Methods

`quicktime.std.clocks.TimeBase.setStopTime()`

See Also

For the corresponding get function, see `GetTimeBaseStopTime` (I–520).

SetTimeBaseTime

Sets the current time of a time base.

```
void SetTimeBaseTime (
    TimeBase      tb,
    const TimeRecord *tr );
```

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II–1119).

SetTimeBaseValue

tr

A pointer to a TimeRecord (IV–2486) structure that contains the current time value.

function result You can access error returns from this function through GetMoviesError (I–492) and GetMoviesStickyError (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Working With Time Base Values” (V–2760)

Related Java Methods

quicktime.std.clocks.TimeBase.setTime()

See Also

For the corresponding get function, see GetTimeBaseTime (I–521).

SetTimeBaseValue

Sets the current time of a time base.

```
void SetTimeBaseValue (
    TimeBase    tb,
    TimeValue    t,
    TimeScale    s );
```

tb

The time base for this operation. Your application obtains this time base identifier from NewTimeBase (II–1119).

t

The new time value.

s

The time scale of the new time value.

function result You can access error returns from this function through GetMoviesError (I–492) and GetMoviesStickyError (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Time Base Values” (V–2760)

Related Java Methods

`quicktime.std.clocks.TimeBase.setValue()`

SetTimeBaseZero

Changes the offset from a time base to either its master time base or its clock component.

```
void SetTimeBaseZero (
    TimeBase      tb,
    TimeRecord    *zero );
```

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (II–1119).

zero

A pointer to the time that corresponds to a 0 time value for the slave time scale. This parameter allows you to set an offset between the time base and its time source. Set this parameter to `NIL` if there is no offset.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

You establish the initial offset when you assign the time base to its time source.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Disposing of Time Bases” (V–2759)

Related Java Methods

`quicktime.std.clocks.TimeBase.setZero()`

SetTrackAlternate

SetTrackAlternate

Adds tracks to, or remove tracks from, alternate groups.

```
void SetTrackAlternate (
    Track    theTrack,
    Track    alternateT );
```

theTrack

The track and group for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497). `SetTrackAlternate` changes this track’s group affiliation based on the value of the `alternateT` parameter.

alternateT

Controls whether the function adds the track to a group or removes it from a group. If this parameter contains a valid track identifier, the Movie Toolbox adds this track to the group that contains the track specified by the parameter `theTrack`. If the track identified by this parameter already belongs to a group, the Movie Toolbox combines the two groups into a single group.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Alternate Tracks” (V–2738)

Related Java Methods

`quicktime.std.movies.Track.setAlternate()`

See Also

For the corresponding get function, see `GetTrackAlternate` (I–522).

SetTrackClipRgn

Sets the clipping region of a track.

```
void SetTrackClipRgn (
    Track    theTrack,
    RgnHandle theClip );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

theClip

A handle to the track’s clipping region. The Movie Toolbox makes a copy of this region. Your application must dispose of the region referred to by this parameter when you are done with it. Set this parameter to `NIL` to disable clipping for the track.

function result

You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Track.setClipRgn()`

See Also

For the corresponding get function, see `GetTrackClipRgn` (I–524).

SetTrackDimensions

Establishes a track’s source rectangle.

```
void SetTrackDimensions (
    Track    theTrack,
    Fixed    width,
    Fixed    height );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

width

A fixed-point number that specifies the width, in pixels, of the track’s rectangle. This value corresponds to the x coordinate of the lower-right corner of the track’s rectangle.

SetTrackEnabled

height

A fixed-point number that specifies the height, in pixels, of the track's rectangle. This value corresponds to the y coordinate of the lower-right corner of the track's rectangle.

function result You can access error returns from this function through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494). See “Error Codes” (IV-2663).

Discussion

If you change the dimensions of an existing track, the media data is scaled to fit into the new rectangle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V-2728)

Related Java Methods

`quicktime.std.movies.Track.setDimensions()`

See Also

For the corresponding get function, see `GetTrackDimensions` (I-527).

SetTrackEnabled

Enables or disables a track.

```
void SetTrackEnabled (
    Track      theTrack,
    Boolean    isEnabled );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II-1092) and `GetMovieTrack` (I-497).

isEnabled

Enables or disables the track. Set this parameter to `TRUE` to enable the track. Set this parameter to `FALSE` to disable the track.

function result You can access error returns from this function through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494). See “Error Codes” (IV-2663).

Discussion

The Movie Toolbox services only enabled tracks.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Disabling Movies and Tracks” (V–2724)

Related Java Methods

`quicktime.std.movies.Track.setEnabled()`

See Also

For the corresponding get function, see `GetTrackEnabled` (I–531).

SetTrackGWorld

Forces a track to draw into a particular graphics world, which may be different from that of the movie.

```
void SetTrackGWorld (
    Track          theTrack,
    CGrafPtr       port,
    GDHandle       gdh,
    TrackTransferUPP proc,
    long           refCon );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

port

Points to the graphics port structure or graphics world to which to draw the track. Set this parameter to `NIL` to use the movie’s graphics port.

gdh

A handle to the movie’s graphics device structure. Set this parameter to `NIL` to use the current device. If the `port` parameter specifies a graphics world, set this parameter to `NIL` to use that graphics world’s graphics device.

proc

A pointer to your `TrackTransferProc` (III–2151) callback. Set this parameter to `NIL` if you want to remove your callback.

SetTrackLayer

refCon

A value to pass to your `TrackTransferProc` callback. Use this parameter to point to a data structure containing any information your callback needs.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

After this function draws a track, it calls your transfer callback to copy the track to the actual movie graphics world. When your transfer callback is called, the current graphics world is set to the correct destination. You can also install a transfer callback and set the graphics world to `NIL`. In this case, the function calls your callback only as a notification that the track has been drawn; no transfer needs to take place.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Track.setGWorld()`

SetTrackLayer

Sets a track’s layer.

```
void SetTrackLayer (
    Track    theTrack,
    short    layer );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

layer

The track’s layer number. Layers are numbered from –32,768 through 32,767. When you create a new track, the Movie Toolbox sets its track number to 0.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Track.setLayer()`

See Also

For the corresponding get function, see `GetTrackLayer` (I–532).

S

SetTrackLoadSettings

Specifies a portion of a track that is to be loaded into memory whenever it is played.

```
void SetTrackLoadSettings (
    Track          theTrack,
    TimeValue      preloadTime,
    TimeValue      preloadDuration,
    long           preloadFlags,
    long           defaultHints );
```

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

`preloadTime`

The starting point of the portion of the track to be preloaded. Set this parameter to –1 if you want to preload the entire track (in this case the function ignores the `preloadDuration` parameter). This parameter should be specified using the movie’s time scale.

`preloadDuration`

The amount of the track to be preloaded, starting from the time specified in the `preloadTime` parameter. If you are preloading the entire track, the function ignores this parameter.

`preloadFlags`

Controls when the toolbox preloads the track. The function supports the following flag values:

`defaultHints`

Specifies playback hints for the track. You may specify any of the supported hints flags.

SetTrackMatrix

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

preloadFlags Constants

`preloadAlways`

Specifies that the toolbox should always preload this track, even if the track is disabled.

`preloadOnlyIfEnabled`

Specifies that the toolbox should preload this track only when the track is enabled.

Discussion

This function allows you to control how the toolbox preloads the tracks in your movie. By using its settings, you make this information part of the movie, so that the preloading takes place every time the movie is opened, without an application having to call `LoadTrackIntoRam` (II–782). Consequently, you should use this feature carefully, so that your movies don’t consume large amounts of memory when opened.

Special Considerations

The toolbox transfers this preload information when you call `CopyTrackSettings` (I–141). In addition, the preload information is preserved when you save or flatten a movie. In flattened movies, the tracks that are to be preloaded are stored at the start of the movie, rather than being interleaved with the rest of the movie data. This improves preload performance.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Enhancing Movie Playback Performance” (V–2722)

Related Java Methods

`quicktime.std.movies.Track.setLoadSettings()`

See Also

For the corresponding get function, see `GetTrackLoadSettings` (I–532).

SetTrackMatrix

Establishes a track’s transformation matrix.

`void SetTrackMatrix (`

```
Track          theTrack,
const MatrixRecord *matrix );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

matrix

A pointer to a `MatrixRecord` (IV–2304) structure that contains the track’s new matrix. If you set this parameter to `NIL`, the Movie Toolbox uses the identity matrix.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

Related Java Methods

`quicktime.std.movies.Track.setMatrix()`

See Also

See `SetIdentityMatrix` (III–1593).

SetTrackMatte

Sets a track’s matte.

```
void SetTrackMatte (
    Track          theTrack,
    PixMapHandle    theMatte );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

theMatte

A handle to the matte. The Movie Toolbox makes a copy of the matte, including its `ColorTable` (IV–2210) structure and pixels. Consequently, your application

SetTrackOffset

must dispose of the matte when you are done with it. Set this parameter to `NIL` to remove the track's matte.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

This matte defines which of the track's pixels are displayed in a movie. You must specify the matte in a `PixelFormat` (IV–2332) structure. The Movie Toolbox displays the weighted average of the track and its destination based on the corresponding pixel in the matte.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Movie Spatial Characteristics” (V–2728)

See Also

For the corresponding get function, see `GetTrackMatte` (I–534).

SetTrackOffset

Modifies the duration of the empty space that lies at the beginning of a track, thus changing the duration of the entire track.

```
void SetTrackOffset (
    Track          theTrack,
    TimeValue      movieOffsetTime );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

movieOffsetTime

The track's offset from the start of the movie, and must be expressed in the time scale of the movie that contains the track.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

All of the tracks in a movie use the movie's time coordinate system. That is, the movie's time scale defines the basic time unit for each of the movie's tracks. Each track begins at the beginning of the movie, but the track's data might not begin until some time value other than 0. This intervening time is represented by blank space. In an audio track the blank space translates to silence; in a video track the blank space generates no visual image. Each track has its own duration. This duration need not correspond to the duration of the movie. Movie duration always equals the maximum duration of all the tracks.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Working With Track Time" (V-2733)

Related Java Methods

`quicktime.std.movies.Track.setOffset()`

See Also

For the corresponding get function, see `GetTrackOffset` (I-539).

SetTrackReference

Modifies an existing track reference.

```
OSErr SetTrackReference (
    Track    theTrack,
    Track    refTrack,
    OSType   refType,
    long     index );
```

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II-1092) and `GetMovieTrack` (I-497).

refTrack

The track to be identified in the track reference. The toolbox uses this information to update the existing track reference.

refType

The type of reference.

SetTrackSoundLocalizationSettings

index

The index value of the reference to be changed. You obtain this index value when you create the track reference.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You may change the track reference so that it identifies a different track in the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Track References” (V–2737)

Related Java Methods

`quicktime.std.movies.Track.setReference()`

See Also

For the corresponding get function, see `GetTrackReference` (I–541).

SetTrackSoundLocalizationSettings

Applies 3D sound effect data to a track.

```
OSErr SetTrackSoundLocalizationSettings (
    Track      theTrack,
    Handle     settings );
```

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

settings

A handle to the settings you want to apply, in the format of an `SSpLocalizationData` (IV–2460) record. You can pass a `NIL` handle to indicate that no 3D sound effects should be used for this track. This function makes a copy of the handle passed, so the caller is responsible for disposing of it.

function result You can access Movie Toolbox error returns through `GetMoviesError`

(I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

This function replaces the 3D sound settings for the specified track with the new `SSpLocalizationData` record contained in the `settings` parameter. The effect of the new 3D sound setting takes place immediately. This call always stores the new record passed, even if the track or the computer is not capable of actually meeting the request. When the movie is saved, the 3D sound settings are saved with it.

The following example code shows how to set the static 3D sound setting for a track using this function:

```
// SetTrackSoundLocalizationSettings coding example
void setTrackSoundLocalization(Track t)
{
    SSpLocalizationData loc;
    Handle h;
    OSErr err;

    loc.cpuLoad = 0;
    loc.medium = kSSpMedium_Air;
    loc.humidity = 0;
    loc.roomSize = 250;
    loc.roomReflectivity = -5;
    loc.reverbAttenuation = -5;
    loc.sourceMode = kSSpSourceMode_Localized;
    loc.referenceDistance = 1;
    loc.coneAngleCos = 0;
    loc.coneAttenuation = 0;
    loc.currentLocation.elevation = 0;
    loc.currentLocation.azimuth = 0;
    loc.currentLocation.distance = 2;
    loc.currentLocation.projectionAngle = 0;
    loc.currentLocation.sourceVelocity = 0;
    loc.currentLocation.listenerVelocity = 0;
    loc.reserved0 = 0;
    loc.reserved1 = 0;
    loc.reserved2 = 0;
    loc.reserved3 = 0;
    loc.virtualSourceCount = 0;
```

SetTrackUsage

```
err = PtrToHand(&loc, &h, sizeof(loc));  
err = SetTrackSoundLocalizationSettings(t, h);  
DisposeHandle(h);  
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Track Sound” (V–2740)

Related Java Methods

`quicktime.std.movies.Track.setSoundLocalizationSettings()`

See Also

For the corresponding get function, see `GetTrackSoundLocalizationSettings` (I–543).

SetTrackUsage

Specifies whether a track is used in a movie, its **preview**, its **poster**, or a combination of these.

```
void SetTrackUsage (  
    Track    theTrack,  
    long     usage );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

usage

Contains flags (see below) that specify how the track is to be used. Be sure to set unused flags to 0.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

usage Constants

`trackUsageInMovie`

If this flag is set to 1, the track is used in the movie.

`trackUsageInPreview`

If this flag is set to 1, the track is used in the **preview**.

trackUsageInPoster

If this flag is set to 1, the track is used in the **poster**.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V–2718)

Related Java Methods

`quicktime.std.movies.Track.setUsage()`

See Also

For the corresponding get function, see `GetTrackUsage` (I–545).

SetTrackVolume

Sets a track’s current volume.

```
void SetTrackVolume (
    Track    theTrack,
    short    volume );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

volume

The current volume setting of the track represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from –1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. You can use constants (see below) for full volume and no volume.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

volume Constants

`kFullVolume`

Sets the track to full volume (constant value is 1.0).

`kNoVolume`

Sets the track to no volume (constant value is 0.0).

SetupAIFFHeader

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Sound Volume” (V–2732)

Related Java Methods

`quicktime.std.movies.Track.setVolume()`

See Also

For the corresponding get function, see `GetTrackVolume` (I–547).

SetupAIFFHeader

Sets up an AIFF or AIFF-C sound file.

```
OSErr SetupAIFFHeader (
    short          fRefNum,
    short          numChannels,
    UnsignedFixed  sampleRate,
    short          sampleSize,
    OSType         compressionType,
    unsigned long  numBytes,
    unsigned long  numFrames );
```

`fRefNum`

A file reference number of a file that is open for writing.

`numChannels`

The number of channels for the sound: 1 for monaural sound and 2 for stereo sound.

`sampleRate`

The rate at which the sound was recorded. To accommodate sample rates greater than 32 kHz, the most significant bit is not treated as a sign bit; instead, that bit is interpreted as having the value 32,768.

`sampleSize`

The sample size for the original sound in bits per sample.

`compressionType`

The compression type for the sound, such as 'NONE', 'MAC3', 'MAC6', or other types. See “Sound Formats” (IV–2692).

numBytes

The number of bytes of sound data that are to be stored in the Common Chunk of the AIFF or AIFF-C file. If recording produces an odd number of bytes of sound data, you must add a pad byte to make the total number of bytes even.

numFrames

The number of sample frames for the sample sound. If you are using a compression type defined by Apple, you can pass 0 in this field and the appropriate value for this field will be computed automatically.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function creates an AIFF or AIFF-C file header, depending on the parameters passed to it. Uncompressed sounds (that is, the `compressionType` parameter is 'NONE') of any type are stored in AIFF format. Compressed sounds of any type are stored in AIFF-C format. The `SetupAIFFHeader` function might format a sound file as an AIFF file even if the File Manager file type of a file is 'AIFC'. The Sound Manager will still play such files correctly.

The AIFF header information is written starting at the current file position of the file specified by the `fRefNum` parameter, and the file position is left at the end of the header upon completion. The `SetupAIFFHeader` function creates a Form Chunk, a Format Version Chunk, a Common Chunk, and a Sound Data chunk, but it does not put any sound data at the end of the Sound Data Chunk.

A good way to use this routine is to create a file that you want to store a sound in, then call `SetupAIFFHeader` with `numBytes` set to 0 to position the file to be ready to write the audio data. Then record the data to the file, set the file position to the beginning of the file, and call `SetupAIFFHeader` again with `numBytes` set to the correct amount of sound data recorded. The file created in this way can be passed to `SndStartFilePlay` (III–1847) to play the sound.

Special Considerations

Because the `SetupAIFFHeader` function moves memory, you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SetupSndHeader

Constructs a sound **resource** containing sampled sound that can be passed to SndPlay (III–1842).

```
OSErr SetupSndHeader (
    SndListHandle    sndHandle,
    short            numChannels,
    UnsignedFixed     sampleRate,
    short            sampleSize,
    OSType            compressionType,
    short            baseNote,
    unsigned long     numBytes,
    short            *headerLen );
```

sndHandle

A handle to a block of memory that is at least large enough to store a SndListResource (IV–2447) structure. The handle is not resized in any way upon successful completion of this call. This function simply fills the relocatable block specified by this parameter with the header information needed for a format 1 'snd ' **resource**, including the sound resource header, the list of sound commands, and a sampled sound header. It is your application’s responsibility to append the desired sampled-sound data.

numChannels

The number of channels for the sound; one channel is equivalent to monaural sound and two channels are equivalent to stereo sound.

sampleRate

The rate at which the sound was recorded. To accommodate sample rates greater than 32 kHz, the most significant bit is not treated as a sign bit; instead, that bit is interpreted as having the value 32,768.

sampleSize

The sample size for the original sound (that is, bits per sample).

compressionType

The compression type for the sound, such as 'NONE', 'MAC3', 'MAC6', or other types. See “Sound Formats” (IV–2692).

baseNote

The base frequency for the sound, expressed as a MIDI note value.

numBytes

The number of bytes of audio data that are to be stored in the handle. This value is not necessarily the same as the number of samples in the sound.

headerLen

On return, the size (in bytes) of the 'snd ' **resource** header that is created. In no case will this length exceed 100 bytes. This field allows you to put the audio data right after the header in the relocatable block specified by the `sndHandle` parameter. The value returned depends on the type of sound header created. Uncompressed sound with a single 8-bit channel goes into a `SoundHeader` (IV–2452) structure. Other uncompressed sound goes into an `ExtSoundHeader` (IV–2255) structure. Compressed sound goes into a `CmpSoundHeader` (IV–2176) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function creates a format 1 'snd ' **resource** for a sampled sound. The resource contains a sound resource header that links the sound to the sampled synthesizer, a single sound command (a `bufferCmd` command to play the accompanying data), and a sampled sound header. You can use this function to construct a `SndListResource` (IV–2447) structure that can be passed to `SndPlay` (III–1842) or stored as an 'snd ' resource. After calling this function, your application should place the sampled-sound data directly after the sampled sound header so that, in essence, the sampled sound header’s final field contains the sound data.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

Related Java Methods

```
quicktime.sound.SndHandle.SndHandle(),
quicktime.sound.SndHandle.setupHeader()
```

SetUserDataItem

Sets an item in a **user data list**.

```
OSErr SetUserDataItem (
    UserData    theUserData,
    void        *data,
    long        size,
    OSType      udType,
    long        index );
```

SFGetFilePreview

theUserData

The **user data list** for this operation. You obtain this item reference by calling GetMovieUserData (I–499), GetTrackUserData (I–546), or GetMediaUserData (I–456).

data

A pointer to the data item to be set in a **user data list**.

size

The size of the information pointed to by the data parameter.

udType

The type value assigned to the new item.

index

The item’s index value. This parameter must specify an item in the **user data list** identified by theUserData. An index value of 0 or 1 implies the first item, which is created if it doesn’t already exist.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Working With Movie User Data” (V–2743)

Related Java Methods

quicktime.std.movies.media.UserData.setDataItem()

See Also

For the corresponding get function, see GetUserDataItem (I–548).

SFGetFilePreview

Displays a file **preview** in an Open dialog box using a standard file reply structure.

```
void SFGetFilePreview (
    Point           where,
    ConstStr255Param prompt,
    FileFilterUPP   fileFilter,
    short           numTypes,
    ConstSFTYPEListPtr typeList,
    DlgHookUPP      dlgHook,
```



```
SFReply          *reply );
```

where

The location of the upper-left corner of the dialog box in global coordinates. If you set this to (-1 -1), the Movie Toolbox centers the dialog box on the main screen. If you set it to (-2 -2), the Movie Toolbox centers the dialog box on the screen that has the best display characteristics.

prompt

This parameter is ignored; it is included for historical reasons only.

fileFilter

Points to a `FileFilterProc` (III–2082) callback that filters the files that are displayed to the user in the dialog box. This is an optional callback provided by your application; if you don't want to supply a filter function, set this parameter to `NIL`. The function uses this parameter along with the `numTypes` and `typeList` parameters to determine which files appear in the dialog box.

numTypes

The number of file types in the array specified by the `typeList` parameter (a number between 1 and 4). Set this parameter to -1 to display all files.

typeList

An array of file types to be displayed to the user. This function only displays files whose type matches an entry in this array (unless you set the `numTypes` parameter to -1; in this case, the function displays all files to the user).

dlgHook

A pointer to an `DlgHookProc` (III–2079) callback. Use this parameter to support a custom dialog box function you have supplied. If you are not supplying a custom dialog box callback, set this parameter to `NIL`.

reply

A pointer to an `SFReply` (IV–2431) structure that is to receive information about the user's selection.

Discussion

This function presents an Open dialog box to the user and allows the user to view file **previews** during the dialog. It automatically converts files to movies if your application requests movies. If a file could be converted into a movie file using a movie import component, then the file is shown in the Standard File dialog box.

Special Considerations

This is the preferred function for displaying a file **preview**.

Version Notes

Introduced in QuickTime 3 or earlier.

SFPGetFilePreview

Programming Info

C interface file: ImageCompression.h

Programming summary: “Displaying File Previews” (V–2758)

SFPGetFilePreview

Displays file **previews** in an Open dialog box using a standard file reply structure.

```
void SFPGetFilePreview (
    Point                where,
    ConstStr255Param     prompt,
    FileFilterUPP        fileFilter,
    short                numTypes,
    ConstSFTYPEListPtr   typeList,
    DlgHookUPP           dlgHook,
    SFReply              *reply,
    short                dlgID,
    ModalFilterUPP       filterProc );
```

where

The location of the upper-left corner of the dialog box in global coordinates. If you set this to (-1 -1), the Movie Toolbox centers the dialog box on the main screen. If you set it to (-2 -2), the Movie Toolbox centers the dialog box on the screen that has the best display characteristics.

prompt

This parameter is ignored; it is included for historical reasons only.

fileFilter

Points to a FileFilterProc (III–2082) callback that filters the files that are displayed to the user in the dialog box. This is an optional callback provided by your application; if you don’t want to supply a filter function, set this parameter to NIL. The function uses this parameter along with the numTypes and typeList parameters to determine which files appear in the dialog box.

numTypes

The number of file types in the array specified by the typeList parameter (a number between 1 and 4). Set this parameter to -1 to display all files.

typeList

An array of file types to be displayed to the user. SFPGetFilePreview only displays files whose type matches an entry in this array (unless you set the numTypes parameter to -1; in this case, the function displays all files to the user).

`dlgHook`

A pointer to an `DlgHookProc` (III–2079) callback. Use this parameter to support a custom dialog box function you have supplied. The dialog template’s **resource** type must be set to `'DLOG'`; you must also supply an item list in a `'DITL'` resource. You specify the dialog template’s resource ID with the `dlgID` parameter. If you are not supplying a custom dialog box callback, set this parameter to `NIL`.

`reply`

A pointer to an `SFReply` (IV–2431) structure that is to receive information about the user’s selection.

`dlgID`

The **resource** ID of your custom dialog template. Use this parameter to specify a custom dialog template resource that has a resource type that differs from the standard value. Set this parameter to 0 to use the standard template.

`filterProc`

A pointer to your `ModalFilterProc` (III–2098) callback. This callback gives you greater control over the interface presented to the user.

Discussion

This function presents an Open dialog box to the user and allows the user to view file **previews** during the dialog. It differs from `SFGetFilePreview` (III–1674) in that you can provide a custom dialog box with any **resource** type and you can specify a modal-dialog filter function that allows you to gain greater control over the user interface. This function automatically converts files to movies if your application requests movies. If a file could be converted into a movie file using a movie import component, then the file is shown in the Standard File dialog box.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Displaying File Previews” (V–2758)

SGAddExtendedFrameReference

Stores extended sample references for a channel component.

```
ComponentResult SGAddExtendedFrameReference (
    SeqGrabComponent      s,
    SeqGrabExtendedFrameInfoPtr  frameInfo );
```

SGAddExtendedMovieData

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

frameInfo

A pointer to a `SeqGrabExtendedFrameInfo` (IV–2429) structure. Your component must place the appropriate information into this structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function differs from `SGAddFrameReference` (III–1681) in that it uses a `SeqGrabExtendedFrameInfo` (IV–2429) structure instead of a `SeqGrabFrameInfo` (IV–2430) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

See Also

See `SGAddFrameReference` (III–1681).

SGAddExtendedMovieData

Adds data to a movie without writing data to a movie file.

```
ComponentResult SGAddExtendedMovieData (
    SeqGrabComponent    s,
    SGChannel           c,
    Ptr                 p,
    long                len,
    wide                *offset,
    long                chRefCon,
    TimeValue           time,
    short               writeType,
    SGOutput             *whichOutput );
```

s
An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III-1751).

c
Identifies the connection to your channel.

p
The location of the data to be added to the movie.

len
The number of bytes of data to be added to the movie.

offset
A pointer to a wide integer that receives the offset to the new data in the movie. If the movie is in memory, the returned offset reflects the location the data will have in the movie on a permanent storage device.

chRefCon
The reference constant for your channel.

time
The time at which the frame was captured, expressed in the time scale associated with your channel.

writeType
A constant (see below) that determines the type of write operation to be used.

whichOutput
The use of `whichOutput` depends on the value passed in the `writeType` parameter. If `writeType` is `seqGrabWriteAppend` or `seqGrabWriteReserve`, the `whichOutput` parameter is a return value specifying the sequence grabber output to which data was written or in which space was reserved. If `writeType` is `seqGrabWriteFill`, the `whichOutput` parameter is an input value indicating which sequence grabber output the data should be written to.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

writeType Constants

seqGrabWriteAppend
Append new data to the end of the file. The sequence grabber sets the field referenced by the `offset` parameter to reflect the location at which data is added.

SGAddFrame

`seqGrabWriteReserve`

Do not write data to the output file, but reserve space in that file for the amount of data specified by the `len` parameter. The sequence grabber sets the field referenced by the `offset` parameter to the location of the reserved space.

`seqGrabWriteFill`

Write the data to the location specified by the field referenced by the `offset` parameter. The sequence grabber sets that field to the location of the byte following the last previously written byte. Use this option to fill space reserved when the `writeType` parameter was set to `seqGrabWriteReserve`.

Discussion

This function differs from `SGAddMovieData` (III–1682) in two respects: the `offset` parameter allows a 64-bit value, and the `whichOutput` parameter does not exist in `SGAddMovieData`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

SGAddFrame

Provides default values for your add-frame function.

```
ComponentResult SGAddFrame (
    SGChannel          c,
    short              bufferNum,
    TimeValue          atTime,
    TimeScale          scale,
    const SGCompressInfo *ci );
```

`c`

The reference that identifies the channel for this operation. The sequence grabber component provides this value to your add-frame function.

`bufferNum`

Identifies the buffer. The sequence grabber component provides this value to your add-frame function.

atTime

The time at which the frame was captured, in the time scale specified by the *scale* parameter. The sequence grabber component provides this value to your add-frame function. Your add-frame function can change this value before calling this function. You can determine the duration of a frame by subtracting its capture time from the capture time of the next frame in the sequence.

scale

The time scale of the movie. The sequence grabber component provides this value to your add-frame function.

ci

A pointer to a `SGCompressInfo` (IV–2432) structure. This structure contains information describing the compression characteristics of the image to be added to the movie.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You should call this function only from your add-frame function. If you call it at any other time, results are unpredictable.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

SGAddFrameReference

Stores sample references for a channel component.

```
ComponentResult SGAddFrameReference (
    SeqGrabComponent      s,
    SeqGrabFrameInfoPtr    frameInfo );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

frameInfo

A pointer to a `SeqGrabFrameInfo` (IV–2430) structure. Your component must completely specify the reference by placing the appropriate information into the record referred to by this parameter.

SGAddMovieData

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

SGAddMovieData

Lets a channel component add data to a movie.

```
ComponentResult SGAddMovieData (
    SeqGrabComponent    s,
    SGChannel            c,
    Ptr                  p,
    long                 len,
    long                 *offset,
    long                 chRefCon,
    TimeValue            time,
    short                 writeType );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through SGInitChannel (III–1751).

c

Identifies the connection to your channel.

p

The location of the data to be added to the movie.

len

Indicates the number of bytes of data to be added to the movie.

offset

A pointer to a field that is to receive the offset to the new data in the movie. The sequence grabber component returns an offset that is correct in the context of the movie **resource**, even if the movie is currently stored in memory. That is, if the movie is in memory, the returned offset reflects the location that the data will have in a movie on a permanent storage device, such as a disk.

`chRefCon`

Your channel's reference constant.

`time`

The time at which your channel captured the frame. This time value is expressed in your channel's time scale.

`writeType`

A constant (see below) that determines the type of write operation to be used.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

writeType Constants

`seqGrabWriteAppend`

Append new data to the end of the file. The sequence grabber sets the field referenced by the `offset` parameter to reflect the location at which data is added.

`seqGrabWriteReserve`

Do not write data to the output file, but reserve space in that file for the amount of data specified by the `len` parameter. The sequence grabber sets the field referenced by the `offset` parameter to the location of the reserved space.

`seqGrabWriteFill`

Write the data to the location specified by the field referenced by the `offset` parameter. The sequence grabber sets that field to the location of the byte following the last previously written byte. Use this option to fill space reserved when the `writeType` parameter was set to `seqGrabWriteReserve`.

Discussion

This function combines the services provided by `SGWriteMovieData` (III–1824) and `SGAddFrameReference` (III–1681). Your channel component should not write data directly to the movie file; use this function instead.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

SGAddOutputDataRefToMedia

Manages capture sessions that involve multiple data references.

SGAlignChannelRect

```
ComponentResult SGAddOutputDataRefToMedia (
    SeqGrabComponent      s,
    SGOutput              sgOut,
    Media                  theMedia,
    SampleDescriptionHandle desc );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

sgOut

A pointer to the current sequence grabber output.

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

desc

A handle to a `SampleDescription` (IV–2414) structure that contains an index, which is assigned to the data by this function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is usually called from the `SGWriteSamples` (III–1825) function of a sequence grabber channel component. You pass to it a sequence grabber output along with a media and `SampleDescription` (IV–2414) structure, and it adds the data reference to the data reference list of the specified media. It also updates the data reference index field of the `SampleDescription` structure to refer to the data reference.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

SGAlignChannelRect

Determines whether or not a channel prefers to draw at a particular screen location.

```
ComponentResult SGAlignChannelRect (
    SGChannel    c,
```

```
Rect      *r );
```

C

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

r

A pointer to a `Rect` (IV–2399) structure. On entry, this structure contains coordinates at which the sequence grabber would like to draw your captured video image. If your component can draw more efficiently or at a higher frame rate at a different location, update the contents of this structure to reflect where you would prefer to draw. The rectangle will be passed in with global, not local, coordinates.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is called by a sequence grabber to determine whether or not a channel prefers to draw at a particular screen location.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828)

SGAppendDeviceListToMenu

Places a list of device names into a specified menu.

```
ComponentResult SGAppendDeviceListToMenu (
    SeqGrabComponent    s,
    SGDeviceList         list,
    MenuHandle           mh );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

list

Defines a pointer to a device list structure pointer. The sequence grabber appends the name of each device in the list to the menu specified by the `mh`

SGChangedSource

parameter. If the `sgDeviceNameFlagDeviceUnavailable` flag is set to 1 for a device in the list, the sequence grabber disables the menu item corresponding to that device.

mh

A handle to the menu to which the device names are to be appended.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Channel Devices” (V–2818)

SGChangedSource

Notifies the sequence grabber that a component is now using a different device.

```
ComponentResult SGChangedSource (  
    SeqGrabComponent    s,  
    SGChannel            c );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

c

Identifies the connection to your channel.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

SGChannelGetCodecSettings

Gets the codec settings for a sequence grabber channel.

```
ComponentResult SGChannelGetCodecSettings (
    SGChannel    c,
    Handle       *settings );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

settings

A pointer to a handle that the codec should resize and fill in with the current internal settings. These settings are codec-defined and usually opaque. Don't dispose of this handle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding set function, see `SGChannelSetCodecSettings` (III–1689). The equivalent function for sound is `SoundComponentGetInfo` (III–1849) with the `siDecompressionParams` selector.

SGChannelGetDataSourceName

Returns the data source name for a track.

```
ComponentResult SGChannelGetDataSourceName (
    SGChannel    c,
    Str255       name,
    ScriptCode    *scriptTag );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

name

A string that is to receive the source identification information. Set this parameter to `NIL` if you do not want to retrieve the name.

SGChannelGetRequestedDataRate

`scriptTag`

A field that is to receive the source information's language code; see “Localization Codes” (IV–2673). Set this parameter to `NIL` if you do not want this information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function allows you to get the source information specified with `SGChannelSetDataSourceName` (III–1690).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

Related Java Methods

`quicktime.std.sg.SGChannel.getDataSourceName()`

See Also

For the corresponding set function, see `SGChannelSetDataSourceName` (III–1690).

SGChannelGetRequestedDataRate

Returns the current maximum data rate requested for a channel.

```
ComponentResult SGChannelGetRequestedDataRate (
    SGChannel      c,
    long           *bytesPerSecond );
```

`c`

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

`bytesPerSecond`

Points to a field that is to receive the maximum data rate requested by the sequence grabber component. This field is set to 0 if the sequence grabber has not set any restrictions.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function allows the sequence grabber component to retrieve the current maximum data rate value from your channel component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

See Also

For the corresponding set function, see `SGChannelSetRequestedDataRate` (III–1691).

SGChannelPutPicture

Undocumented

```
ComponentResult SGChannelPutPicture (
    SGChannel    c );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

SGChannelSetCodecSettings

Sets the codec settings for a sequence grabber channel.

```
ComponentResult SGChannelSetCodecSettings (
    SGChannel    c,
    Handle        settings );
```

SGChannelSetDataSourceName

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

settings

Undocumented

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding get function, see `SGChannelGetCodecSettings` (III–1686).

SGChannelSetDataSourceName

Sets the data source name for a track.

```
ComponentResult SGChannelSetDataSourceName (
    SGChannel          c,
    ConstStr255Param   name,
    ScriptCode          scriptTag );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

name

A string that contains the source identification information. The source information identifies the source of the video data (for example, a videotape name).

scriptTag

The language of the source identification information; see “Localization Codes” (IV–2673).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function allows you to set the source information for a sequence grabber channel. You must set this information before you start digitizing. The sequence grabber channel stores this information in a timecode track in the movie created

after the capture is complete. If the video digitizer does not provide timecode information, the sequence grabber does not save this information.

Special Considerations

This function is currently supported only by video channels.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

Related Java Methods

`quicktime.std.sg.SGChannel.setDataSourceName()`

See Also

For the corresponding get function, see `SGChannelGetDataSourceName` (III–1687).

SGChannelSetRequestedDataRate

Specifies the maximum requested data rate for a channel.

```
ComponentResult SGChannelSetRequestedDataRate (
    SGChannel      c,
    long           bytesPerSecond );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

bytesPerSecond

The maximum data rate requested by the sequence grabber component, in bytes per second. The sequence grabber component sets this parameter to 0 to remove any data-rate restrictions.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function allows the sequence grabber component to specify the maximum rate at which it would like to receive data from your channel component. The data rate supplied by the sequence grabber component represents a requested data rate; your component may not be able to observe that rate under all conditions.

SGCompressFrame

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

See Also

For the corresponding get function, see `SGChannelGetRequestedDataRate` (III–1688).

SGCompressFrame

Provides the default behavior for your compress function.

```
ComponentResult SGCompressFrame (  
    SGChannel    c,  
    short        bufferNum );
```

c

The reference that identifies the channel for this operation. The sequence grabber provides this value to your compress function.

bufferNum

The buffer. The sequence grabber provides this value to your compress function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

You should call this function only from your compress function. If you call it at any other time, results are unpredictable.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

SGCompressFrameComplete

Provides the default behavior for your compress-complete function.

```
ComponentResult SGCompressFrameComplete (  
    SGChannel    c,
```

```
short          bufferNum,
Boolean        *done,
SGCompressInfo *ci );
```

c

The reference that identifies the channel for this operation. The sequence grabber component provides this value to your compress-complete function.

bufferNum

Identifies the buffer. The sequence grabber component provides this value to your compress-complete function.

done

A pointer to a Boolean value. The function sets this Boolean value to TRUE if the compression is complete, or FALSE if the operation is incomplete. The sequence grabber component provides this pointer to your compress-complete function.

ci

A pointer to a SGCompressInfo (IV–2432) structure. If the compression is complete, the function completely formats this structure with information that is appropriate to the frame just compressed. The sequence grabber component provides this pointer to your compress-complete function.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Special Considerations

You should call this function only from your compress-complete function. If you call it at any other time, results are unpredictable.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

SGDisplayCompress

Provides the default behavior for your display-compress function.

```
ComponentResult SGDisplayCompress (
    SGChannel      c,
    Ptr            dataPtr,
    ImageDescriptionHandle desc,
    MatrixRecord    *mp,
    RgnHandle       clipRgn );
```

SGDisplayFrame

c

Identifies the channel for this operation. The sequence grabber provides this value to your display-compress function.

dataPtr

A pointer to the compressed image data. The sequence grabber provides this pointer to your display-compress function.

desc

A handle to the `ImageDescription` (IV–2285) structure to use for the decompression operation. The sequence grabber provides this handle to your display-compress function.

mp

A pointer to a `MatrixRecord` (IV–2304) structure. This structure contains the transformation matrix to use when displaying the image. If there is no matrix for the operation, set this parameter to `NIL`.

clipRgn

A handle to a `MacRegion` (IV–2303) structure that defines the clipping region for the destination image. This region is defined in the destination coordinate system. If there is no clipping region, set this parameter to `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

You should call this function only from your display-compress function. If you call it at any other time, results are unpredictable.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

SGDisplayFrame

Provides the default behavior for your display function.

```
ComponentResult SGDisplayFrame (
    SGChannel      c,
    short          bufferNum,
    const MatrixRecord *mp,
    RgnHandle      clipRgn );
```

c The reference that identifies the channel for this operation. The sequence grabber component provides this value to your display function.

bufferNum Identifies the buffer. The sequence grabber component provides this value to your display function.

mp A pointer to a `MatrixRecord` (IV–2304) structure for the display operation. If there is no matrix for the operation, set this parameter to `NIL`.

clipRgn A handle to a `MacRegion` (IV–2303) structure that defines the clipping region for the destination image. This region is defined in the destination coordinate system. If there is no clipping region, set this parameter to `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

You should call this function only from your display function. If you call it at any other time, results are unpredictable.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

SGDisposeChannel

Removes a channel from a sequence grabber component.

```
ComponentResult SGDisposeChannel (
    SeqGrabComponent s,
    SGChannel c );
```

s The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

c The reference that identifies the channel you want to close. You obtain this reference from `SGNewChannel` (III–1753).

SGDisposeDeviceList

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Configuring Sequence Grabber Components” (V–2809)

Related Java Methods

quicktime.std.sg.SequenceGrabber.disposeChannel()

SGDisposeDeviceList

Disposes of a device list.

```
ComponentResult SGDisposeDeviceList (
    SeqGrabComponent s,
    SGDeviceList list );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from OpenDefaultComponent (II–1131) or OpenComponent (II–1130).

list

A pointer to a pointer to an SGDeviceListRecord (IV–2433) structure. The sequence grabber disposes of the memory used by this structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Working With Channel Devices” (V–2818)

SGDisposeOutput

Disposes of an existing sequence grabber output.

```
ComponentResult SGDisposeOutput (
    SeqGrabComponent s,
    SGOutput sgOut );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

sgOut

Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (III–1757).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Use this function to dispose of an existing output. If any sequence grabber channels are using this output, the sequence grabber component assigns them to an undefined output.

Special Considerations

You cannot dispose of an output when the sequence grabber component is in record mode.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Outputs” (V–2814)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.disposeOutput()`

See Also

See `SGNewOutput` (III–1757).

SGGetAdditionalSoundRates

Returns the additional sound sample rates added to a specified sequence grabber sound channel.

```
ComponentResult SGGetAdditionalSoundRates (
    SGChannel    c,
    Handle       *rates );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

rates

A pointer to the handle where the list of additional sample rates should be returned. If no additional sample rates have been set, this function sets the rates

SGGetAlignmentProc

handle to NIL. The caller of this routine is responsible for disposing of the returned handle.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

See Also

For the corresponding set function, see SGSetAdditionalSoundRates (III–1774).

SGGetAlignmentProc

Obtains information about the best screen positions for a sequence grabber’s video image in terms of appearance and maximum performance.

```
ComponentResult SGGetAlignmentProc (
    SeqGrabComponent      s,
    ICMAAlignmentProcRecordPtr alignmentProc );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from OpenDefaultComponent (II–1131) or OpenComponent (II–1130).

alignmentProc

A pointer to an ICMAAlignmentProcRecord (IV–2278) structure. The sequence grabber places its alignment information into this structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Configuring Sequence Grabber Components” (V–2809)

SGGetBufferInfo

Obtains information about a buffer that has been passed to a callback function.

```
ComponentResult SGGetBufferInfo (
    SGChannel      c,
    short          bufferNum,
    PixMapHandle   *bufferPM,
    Rect           *bufferRect,
    GWorldPtr      *compressBuffer,
    Rect           *compressBufferRect );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

bufferNum

Identifies the buffer. The sequence grabber component provides this value to your callback function.

bufferPM

A pointer to a location that is to receive a handle to the `PixMap` (IV–2332) structure that contains the image. Note that this structure may be offscreen. Do not dispose of this structure. If you do not want this information, set this parameter to `NIL`.

bufferRect

A pointer to a `Rect` (IV–2399) structure that is to receive the dimensions of the image's boundary rectangle. If you do not want this information, set this parameter to `NIL`.

compressBuffer

A pointer to a location that is to receive a pointer to the filter buffer for the image. The sequence grabber component returns this information only if your application has assigned a filter buffer to this video channel. You assign a filter buffer by calling `SGSetCompressBuffer` (III–1784). Do not dispose of this buffer.

compressBufferRect

A pointer to a `Rect` (IV–2399) structure that is to receive the dimensions of the filter buffer for the image. The sequence grabber component returns this information only if your application has assigned a filter buffer to this video channel. You assign a filter buffer by calling `SGSetCompressBuffer` (III–1784). If you have not assigned a filter buffer, the sequence grabber component returns an empty rectangle. If you do not want this information, set this parameter to `NIL`.

SGGetChannelBounds

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

SGGetChannelBounds

Determines a channel’s display boundary rectangle.

```
ComponentResult SGGetChannelBounds (
    SGChannel    c,
    Rect         *bounds );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

bounds

A pointer to a `Rect` (IV–2399) structure that is to receive information about your channel’s display boundary rectangle.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Channel Characteristics” (V–2815)

Related Java Methods

`quicktime.std.sg.VisualChannel.getBounds()`

See Also

For the corresponding set function, see `SGSetChannelBounds` (III–1774).

SGGetChannelClip

Retrieves a channel’s clipping region.

```
ComponentResult SGGetChannelClip (
```

```

    SGChannel      c,
    RgnHandle      *theClip );

```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

theClip

A pointer to a handle that is to receive a `MacRegion` (IV–2303) structure that defines the clipping region. The application is responsible for disposing of this handle. If there is no clipping region, set this handle to `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

Note that some devices may not support clipping.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

Related Java Methods

```

quicktime.qd.Region.fromVideoChannel(),
quicktime.std.sg.VisualChannel.getClip()

```

See Also

For the corresponding set function, see `SGSetChannelClip` (III–1775).

SGGetChannelDeviceList

Retrieves a list of the devices that are valid for a specified channel.

```

ComponentResult SGGetChannelDeviceList (
    SGChannel      c,
    long           selectionFlags,
    SGDeviceList    *list );

```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

SGGetChannelInfo

`selectionFlags`

Flags (see below) that control the data you are to return for each device.

`list`

A pointer to an `SGDeviceListRecord` (IV–2433) structure. The channel creates this structure and returns a pointer to it in the field referred to by this parameter. Applications use `SGDisposeDeviceList` (III–1696) to dispose of the memory used by the list.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

selectionFlags Constants

`sgDeviceListWithIcons`

If you set this flag to 1, the sequence grabber returns an icon for each device in the list, in the `icon` field. If you set this flag to 0, the sequence grabber sets the `icon` field to 0.

`sgDeviceListDontCheckAvailability`

If you set this flag to 1, the sequence grabber does not check the availability of each device. Otherwise, the sequence grabber checks each device’s availability, and sets the `sgDeviceNameFlagDeviceUnavailable` flag appropriately in each `SGDeviceName` (IV–2434) structure that is returned. Note that checking device availability slows this function. In general, however, you should check availability if you plan to present a list of devices to the user. Otherwise, the user may select a device that is unavailable.

Discussion

This function can be useful for retrieving the name of the current device. Retrieve the device list and use the `selectedIndex` field to determine which device is currently in use.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing Channel Devices” (V–2828), “Working With Channel Devices” (V–2818)

SGGetChannelInfo

Determines how a channel’s data is represented to the user: as visual data or audio data, or both.

`ComponentResult` `SGGetChannelInfo` (

```

    SGChannel    c,
    long         *channelInfo );

```

C

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

channelInfo

A pointer to a long integer that is to receive channel information flags (see below). You may set more than one flag to 1. Set unused flags to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

channelInfo Constants

`seqGrabHasBounds`

If you set this flag to 1, the channel has a visual representation. The sequence grabber component may call your `SGSetChannelBounds` (III–1774) function.

`seqGrabHasVolume`

If you set this flag to 1, the channel has an audio representation. The sequence grabber component may call your `SGSetChannelVolume` (III–1783) function.

`seqGrabHasDiscreteSamples`

If you set this flag to 1, the sequence grabber component may use `SGSetChannelMaxFrames` (III–1777) to limit the number of frames processed in a record operation or the rate at which those frames are processed. If your channel’s data is not organized into frames, set this flag to 0.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

SGGetChannelMatrix

Retrieves a channel’s display transformation matrix.

```

ComponentResult SGGetChannelMatrix (
    SGChannel    c,
    MatrixRecord *m );

```

SGGetChannelMaxFrames

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

m

A pointer to a `MatrixRecord` (IV–2304) structure. Place your current matrix values into this structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

Related Java Methods

`quicktime.std.sg.VisualChannel.getMatrix()`

See Also

For the corresponding set function, see `SGSetChannelMatrix` (III–1776).

SGGetChannelMaxFrames

Determines the number of frames left to be captured from a specified channel.

```
ComponentResult SGGetChannelMaxFrames (
    SGChannel      c,
    long           *frameCount );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

frameCount

A pointer to a long integer that is to receive a value specifying the number of frames left to be captured during the **preview** or record operation. If the returned value is `–1`, the sequence grabber channel component captures as many frames as it can.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

See Also

For the corresponding set function, see `SGSetChannelMaxFrames` (III–1777).

SGGetChannelPlayFlags

Retrieves the playback control flags that you set with `SGSetChannelPlayFlags` (III–1779).

```
ComponentResult SGGetChannelPlayFlags (
    SGChannel    c,
    long         *playFlags );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

playFlags

A pointer to a long integer that is to receive flags (see below) that influence channel playback. Set unused flags to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

playFlags Constants

`channelPlayNormal`

Your channel component uses its default playback methodology.

`channelPlayFast`

Your channel component sacrifices playback quality to achieve the specified playback rate.

`channelPlayHighQuality`

Your channel component plays the channel’s data at the highest possible quality. This option sacrifices playback rate for the sake of image quality. It may reduce the amount of processor time available to other programs in the computer. This option should not affect the quality of the recorded data, however.

`channelPlayAllData`

Your channel component tries to play all of the data it captures, even the data that is stored in offscreen buffers. This option is useful when you want to be

SGGetChannelSampleDescription

sure that the user sees as much of the captured data as possible. The sequence grabber component sets this flag to 1 to play all the captured data. The sequence grabber component may combine this flag with any of the other values for the `playFlags` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

Related Java Methods

`quicktime.std.sg.SGChannel.getPlayFlags()`

See Also

For the corresponding set function, see `SGSetChannelPlayFlags` (III–1779).

SGGetChannelSampleDescription

Retrieves a channel’s sample description structure.

```
ComponentResult SGGetChannelSampleDescription (
    SGChannel      c,
    Handle          sampleDesc );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

sampleDesc

A handle that is to receive the structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The channel returns a structure that is appropriate to the type of data being captured. For video channels, the channel component returns an `ImageDescription` (IV–2285) structure; for sound channels, it receives a `SoundDescription` (IV–2449) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

SGGetChannelSettings

Retrieves the current settings of a channel used by the sequence grabber.

```
ComponentResult SGGetChannelSettings (
    SeqGrabComponent  s,
    SGChannel         c,
    UserData          *ud,
    long              flags );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

ud

On return, a pointer to a `UserDataRecord` (IV–2496) structure that contains the configuration information.

flags

Reserved for Apple. Set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Settings” (V–2812)

Related Java Methods

```
quicktime.std.movies.media.UserData.fromSGChannel(),
quicktime.std.sg.SGChannel.getSettings()
```

See Also

For the corresponding set function, see `SGSetChannelSettings` (III–1781).

SGGetChannelTimeBase

SGGetChannelTimeBase

Retrieves a reference to the time base that is being used by a sequence grabber channel.

```
ComponentResult SGGetChannelTimeBase (  
    SGChannel    c,  
    TimeBase     *tb );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

tb

A pointer to a time base identifier, such as that returned by `NewTimeBase` (II–1119).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

SGGetChannelTimeScale

Lets the sequence grabber retrieve a channel’s time scale.

```
ComponentResult SGGetChannelTimeScale (  
    SGChannel    c,  
    TimeScale     *scale );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

scale

A pointer to a time scale. Your channel component places information about its time scale into this structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The time scale you return typically corresponds to the time scale of the media that has been created by your channel. Applications may use this time scale in their data functions.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

S

SGGetChannelUsage

Determines how the sequence grabber component is using a channel.

```
ComponentResult SGGetChannelUsage (
    SGChannel    c,
    long         *usage );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

usage

A pointer to a location that is to receive flags (see below) that specify how your channel is to be used. You may set more than one of these flags to 1. Set unused flags to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

usage Constants

`seqGrabRecord`

This flag is set to 1 if your channel is being used for recording.

`seqGrabPreview`

This flag is set to 1 if your channel is being used for **previewing**.

`seqGrabPlayDuringRecord`

This flag is set to 1 if your channel plays its captured data during a record operation.

Version Notes

Introduced in QuickTime 3 or earlier.

SGGetChannelVolume

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

Related Java Methods

`quicktime.std.sg.SGChannel.getUsage()`

See Also

For the corresponding set function, see `SGSetChannelUsage` (III–1782).

SGGetChannelVolume

Determines a channel’s sound volume setting.

```
ComponentResult SGGetChannelVolume (  
    SGChannel    c,  
    short        *volume );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

volume

A pointer to an integer that is to receive the volume setting of the channel represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from –1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Sound Channel Components” (V–2831), “Working With Channel Characteristics” (V–2815)

Related Java Methods

`quicktime.std.sg.AudioChannel.getVolume()`

See Also

For the corresponding set function, see `SGSetChannelVolume` (III–1783).

SGGetCompressBuffer

Returns information about the filter buffer established for a video channel.

```
ComponentResult SGGetCompressBuffer (
    SGChannel    c,
    short        *depth,
    Rect         *compressSize );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

depth

A pointer to a field that is to receive the pixel depth of the filter buffer. If your component is not filtering the input video data, set the returned value to 0.

compressSize

A pointer to a `Rect` (IV–2399) structure that is to receive the dimensions of the filter buffer. If your component is not filtering the input video data, return an empty rectangle (all coordinates set to 0).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

See Also

For the corresponding set function, see `SGSetCompressBuffer` (III–1784).

SGGetDataOutput

Determines the movie file that is currently assigned to a sequence grabber component and the control flags that would govern a record operation.

```
ComponentResult SGGetDataOutput (
    SeqGrabComponent s,
    FSSpec           *movieFile,
    long             *whereFlags );
```

SGGetDataOutput

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

movieFile

A pointer to an `FSSpec` (IV–2262) structure that is to receive information about the movie file for this record operation.

whereFlags

A pointer to a long integer that is to receive flags (see below) that control the record operation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

whereFlags Constants

seqGrabToDisk

Instructs the sequence grabber component to write the recorded data to a QuickTime movie in the movie file specified by the `movieFile` parameter. If this flag is set to 1, the sequence grabber writes the data to the file as the data is recorded.

seqGrabToMemory

Instructs the sequence grabber component to store the recorded data in memory until the recording process is complete. The sequence grabber then writes the recorded data to the movie file specified by the `movieFile` parameter. This technique provides better performance than recording directly to the movie file, but it limits the amount of data you can record. If this flag is set to 1, the sequence grabber component is recording to memory.

seqGrabDontUseTempMemory

Prevents the sequence grabber component from using temporary memory during the record operation. By default, the sequence grabber component and its channel components use as much temporary memory as necessary to perform the record operation. If this flag is set to 1, the sequence grabber component and its channel components do not use temporary memory.

seqGrabAppendToFile

Directs the sequence grabber component to add the recorded data to the data fork of the movie file specified by the `movieFile` parameter. By default, the sequence grabber component deletes the movie file and creates a new file containing one movie and its movie **resource**. If this flag is set to 1, the sequence grabber component appends the recorded data to the data fork of the movie file and creates a new movie resource in that file.

`seqGrabDontAddMovieResource`

Prevents the sequence grabber component from adding the new movie **resource** to the movie file specified by the `movieFile` parameter. By default, the sequence grabber component creates a new movie resource and adds that resource to the movie file. If this flag is set to 1, the sequence grabber component does not add the movie resource to the movie file. You are then responsible for adding the resource to a file, if you so desire

`seqGrabDontMakeMovie`

Prevents the sequence grabber component from creating a movie. By default, the sequence grabber component creates a new movie **resource** and adds the captured data to that movie. If this flag is set to 1, the sequence grabber still calls your data function, but does not write any data to the movie file.

Discussion

You set the characteristics returned by this function by calling `SGSetDataOutput` (III–1785). If you have not set these characteristics before calling this function, the returned data is meaningless.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Sequence Grabber Components” (V–2809)

Related Java Methods

```
quicktime.io.QTFile.fromSequenceGrabber(),
quicktime.std.sg.SequenceGrabber.getDataOutputFile(),
quicktime.std.sg.SequenceGrabber.getDataOutputFlags()
```

See Also

See `SGSetDataOutput` (III–1785).

SGGetDataOutputStorageSpaceRemaining

Returns the amount of space remaining in the data reference associated with an output.

```
ComponentResult SGGetDataOutputStorageSpaceRemaining (
    SeqGrabComponent    s,
    SGOutput             sgOut,
    unsigned long        *space );
```

SGGetDataOutputStorageSpaceRemaining64

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

sgOut

Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (III–1757).

space

A pointer to an unsigned long integer, where the sequence grabber component returns a value that indicates the number of bytes of space remaining in the data reference associated with the output.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Use this function in place of `SGGetStorageSpaceRemaining` (III–1736) in cases where you are working with more than one output.

Version Notes

A sequence grabber output ties a sequence grabber channel to a specified data reference for the output of captured data. If you are capturing to a single movie file, you can continue to use `SGSetDataOutput` (III–1785) or `SGSetDataRef` (III–1787) to specify the sequence grabber’s destination. However, if you want to capture movie data into several different files or data references, you must use sequence grabber outputs to do so. Even if you are using outputs, you must still use `SGSetDataOutput` or `SGSetDataRef` to identify where the sequence grabber should create the movie **resource**. You are responsible for creating outputs, assigning them to sequence grabber channels, and disposing of them when you are done.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Outputs” (V–2814)

Related Java Methods

`quicktime.std.sg.SGOutput.getDataStorageSpaceRemaining()`

SGGetDataOutputStorageSpaceRemaining64

Provides a 64-bit version of `SGGetDataOutputStorageSpaceRemaining` (III–1713).

```
ComponentResult SGGetDataOutputStorageSpaceRemaining64 (
    SeqGrabComponent s,
    SGOutput          sgOut,
```



```
wide                *space );
```

S

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

sgOut

Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (III–1757).

space

A pointer to a 64-bit wide integer, where the sequence grabber component returns a value that indicates the number of bytes of space remaining in the data reference associated with the output.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Outputs” (V–2814)

See Also

For further information, see `SGGetDataOutputStorageSpaceRemaining` (III–1713).

SGGetDataRate

Determines for a sequence grabber how much recording time is left.

```
ComponentResult SGGetDataRate (
    SGChannel    c,
    long         *bytesPerSecond );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

bytesPerSecond

A pointer to a long integer that is to receive a value indicating the number of bytes your component is recording per second. Your component calculates this value based on its current operational parameters.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

SGGetDataRef

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826)

SGGetDataRef

Determines the data reference currently assigned to a sequence grabber component and the control flags that would govern a record operation.

```
ComponentResult SGGetDataRef (
    SeqGrabComponent s,
    Handle *dataRef,
    OSType *dataRefType,
    long *whereFlags );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

dataRef

A pointer to a handle that is to receive the information that identifies the destination container.

dataRefType

A pointer to a field that is to receive the type of data reference.

whereFlags

A pointer to a long integer that is to receive flags (see below) that control the record operation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

whereFlags Constants

`seqGrabToDisk`

Instructs the sequence grabber component to write the recorded data to a QuickTime movie in the movie file specified by the `movieFile` parameter. If this flag is set to 1, the sequence grabber writes the data to the file as the data is recorded.

`seqGrabToMemory`

Instructs the sequence grabber component to store the recorded data in memory until the recording process is complete. The sequence grabber then writes the recorded data to the movie file specified by the `movieFile` parameter. This technique provides better performance than recording directly to the movie file, but it limits the amount of data you can record. If this flag is set to 1, the sequence grabber component is recording to memory.

`seqGrabDontUseTempMemory`

Prevents the sequence grabber component from using temporary memory during the record operation. By default, the sequence grabber component and its channel components use as much temporary memory as necessary to perform the record operation. If this flag is set to 1, the sequence grabber component and its channel components do not use temporary memory.

`seqGrabAppendToFile`

Directs the sequence grabber component to add the recorded data to the data fork of the movie file specified by the `movieFile` parameter. By default, the sequence grabber component deletes the movie file and creates a new file containing one movie and its movie **resource**. If this flag is set to 1, the sequence grabber component appends the recorded data to the data fork of the movie file and creates a new movie resource in that file.

`seqGrabDontAddMovieResource`

Prevents the sequence grabber component from adding the new movie **resource** to the movie file specified by the `movieFile` parameter. By default, the sequence grabber component creates a new movie resource and adds that resource to the movie file. If this flag is set to 1, the sequence grabber component does not add the movie resource to the movie file. You are then responsible for adding the resource to a file, if you so desire.

`seqGrabDontMakeMovie`

Prevents the sequence grabber component from creating a movie. By default, the sequence grabber component creates a new movie **resource** and adds the captured data to that movie. If this flag is set to 1, the sequence grabber still calls your data function, but does not write any data to the movie file.

Discussion

This function allows you to determine the data reference that is currently assigned to a sequence grabber component and the control flags that would govern a record operation. You set these characteristics by calling `SGSetDataRef` (III–1787). If you have not set these characteristics before calling this function, the returned data is meaningless.

SGGetFlags

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Sequence Grabber Components” (V–2809)

Related Java Methods

```
quicktime.std.sg.SequenceGrabber.getDataRef(),  
quicktime.std.sg.SequenceGrabber.getDataRefFlags(),  
quicktime.std.movies.media.DataRef.fromSequenceGrabber()
```

See Also

For the corresponding set function, see `SGSetDataRef` (III–1787).

SGGetFlags

Retrieves a sequence grabber’s control flags.

```
ComponentResult SGGetFlags (  
    SeqGrabComponent    s,  
    long                 *sgFlags );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

sgFlags

A pointer to a long integer that is to receive the control flag (see below) for the current operation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

sgFlags Constant

`sgFlagControlledGrab`

Informs the sequence grabber component that you are working with a frame-addressable device to perform a controlled record operation. The sequence grabber and its channel components optimize their operation for this situation. This flag allows the sequence grabber component to trade off speed and quality. This flag is set to 1 if you are performing a controlled grab using a frame-addressable source device.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Characteristics” (V–2813)

See Also

For the corresponding set function, see `SGSetFlags` (III–1789).

SGGetFrameRate

Retrieves a video channel’s frame rate for recording.

```
ComponentResult SGGetFrameRate (
    SGChannel      c,
    Fixed          *frameRate );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

frameRate

A pointer to a field to receive the current frame rate.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

See Also

For the corresponding set function, see `SGSetFrameRate` (III–1792).

SGGetGWorld

Determines the graphics port and device for a sequence grabber component.

```
ComponentResult SGGetGWorld (
    SeqGrabComponent  s,
    CGrafPtr          *gp,
    GDHandle           *gd );
```

SGGetIndChannel

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

gp

A pointer to a location that is to receive a pointer to the destination graphics port. Set this parameter to `NIL` if you are not interested in this information.

gd

A pointer to a location that is to receive a handle to the destination graphics device. Set this parameter to `NIL` if you are not interested in this information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Sequence Grabber Components” (V–2809)

Related Java Methods

```
quicktime.qd.QDGraphics.fromSequenceGrabber(),
quicktime.qd.GDevice.fromSequenceGrabber(),
quicktime.std.sg.SequenceGrabber.getGWorld(),
quicktime.std.sg.SequenceGrabber.getGWorldDevice()
```

See Also

For the corresponding set function, see `SGSetGWorld` (III–1792).

SGGetIndChannel

Collects information about all of the channel components currently in use by a sequence grabber component.

```
ComponentResult SGGetIndChannel (
    SeqGrabComponent    s,
    short                index,
    SGChannel            *ref,
    OSType               *chanType );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

index

Specifies an index value that identifies the channel to be queried. The first channel has an index value of 1.

ref

A pointer to a field to receive a value identifying your connection to the channel. If you do not want to receive this information, set this parameter to `NIL`.

chanType

A pointer to a field to receive the channel’s subtype value (see below). This value indicates the media type supported by the channel component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

chanType Constants

`VideoMediaType`

Video channel.

`SoundMediaType`

Sound channel.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Sequence Grabber Components” (V–2809)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.getIndChannelType()`,

`quicktime.std.sg.SequenceGrabber.getIndChannel()`

SGGetInstrument

Gets a tone description for a music sequence grabber channel.

```
ComponentResult SGGetInstrument (
    SGChannel      c,
    ToneDescription *td );
```

SGGetLastMovieResID

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

td

Pointer to a `ToneDescription` (IV–2487) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Related Java Methods

`quicktime.std.sg.SGMusicChannel.getInstrument()`

See Also

For the corresponding set function, see `SGSetInstrument` (III–1794).

SGGetLastMovieResID

Retrieves the last **resource** ID used by the sequence grabber component.

```
ComponentResult SGGetLastMovieResID (
    SeqGrabComponent s,
    short *resID );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

resID

A pointer to an integer that is to receive the **resource** ID the sequence grabber assigned to the movie resource it just created.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Components” (V–2811)

SGGetMaximumRecordTime

Determines the time limit you have set for a record operation.

```
ComponentResult SGGetMaximumRecordTime (
    SeqGrabComponent    s,
    unsigned long        *ticks );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

ticks

A pointer to a long integer that is to receive a value indicating the maximum duration for the record operation, in system **ticks** (sixtieths of a second). A value of 0 indicates that there is no time limit.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Characteristics” (V–2813)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.getMaximumRecordTime()`

See Also

For the corresponding set function, see `SGSetMaximumRecordTime` (III–1796).

SGGetMode

Determines whether a sequence grabber component is in **preview** mode or record mode.

```
ComponentResult SGGetMode (
    SeqGrabComponent    s,
    Boolean              *previewMode,
    Boolean              *recordMode );
```

SGGetMovie

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

`previewMode`

A pointer to a Boolean. The sequence grabber component sets this field to `TRUE` if the component is in **preview** mode.

`recordMode`

A pointer to a Boolean. The sequence grabber component sets this field to `TRUE` if the component is in record mode.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.isPreviewMode()`,

`quicktime.std.sg.SequenceGrabber.isRecordMode()`

SGGetMovie

Returns a reference to the movie that contains the data collected during a record operation.

```
Movie SGGetMovie (
    SeqGrabComponent s );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

function result A movie identifier, such as that returned from `NewMovie` (II–1069).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

`quicktime.std.movies.Movie.fromSequenceGrabber()`,

`quicktime.std.sg.SequenceGrabber.getMovie()`

SGGetNextExtendedFrameReference

Allows a channel component to retrieve the sample references stored previously by `SGAddExtendedMovieData` (III–1678) or `SGAddExtendedFrameReference` (III–1677).

```
ComponentResult SGGetNextExtendedFrameReference (
    SeqGrabComponent          S,
    SeqGrabExtendedFrameInfoPtr frameInfo,
    TimeValue                 *frameDuration,
    long                       *frameNumber );
```

S

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

frameInfo

A pointer to a `SeqGrabExtendedFrameInfo` (IV–2429) structure. Your component must place the appropriate information into this structure.

frameDuration

A pointer to a time value. The sequence grabber component calculates the duration of the specified frame and returns that duration in this structure. The sequence grabber component cannot calculate the duration of the last frame in a sequence. For the last frame, the time value is set to –1.

frameNumber

A pointer to a long integer representing the frame number. Frame numbers need not be sequential, and need not start at 0. To retrieve information about the first frame in a movie, set the integer to –1.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your channel component can process frame references sequentially or randomly. You can specify any relative frame for which you want to retrieve information.

SGGetNextFrameReference

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

See Also

See `SGGetNextFrameReference` (III–1726). This function differs from `SGGetNextFrameReference` in that it fills out a `SeqGrabExtendedFrameInfo` (IV–2429) structure instead of a `SeqGrabFrameInfo` (IV–2430) structure.

SGGetNextFrameReference

Lets a channel component retrieve the sample references that were stored by calling `SGAddMovieData` (III–1682) or `SGAddFrameReference` (III–1681).

```
ComponentResult SGGetNextFrameReference (
    SeqGrabComponent      s,
    SeqGrabFrameInfoPtr    frameInfo,
    TimeValue              *frameDuration,
    long                   *frameNumber );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

frameInfo

A pointer to a `SeqGrabFrameInfo` (IV–2430) structure. Your component must identify itself to the sequence grabber component by setting the `frameChannel` field of this structure to the component instance that identifies the current connection to your channel. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756). The sequence grabber component then returns information about the specified frame in the remaining fields of this structure.

frameDuration

A pointer to a time value. The sequence grabber component calculates the duration of the specified frame and returns that duration in the structure referred to by this parameter. The sequence grabber component cannot calculate the duration of the last frame in a sequence. In this case, the sequence grabber component sets the returned time value to –1.

frameNumber

A pointer to a long integer. Your channel component specifies the frame number corresponding to the frame about which you want to retrieve information. Frames are numbered starting at 0. However, frame numbers need not start at 0, and they need not be sequential. Set the integer to -1 to retrieve information about the first frame in a movie.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V-2833)

SGGetOutputDataReference

Returns information about the data reference associated with the specified sequence grabber output.

```
ComponentResult SGGetOutputDataReference (
    SeqGrabComponent s,
    SGOutput          sgOut,
    Handle             *dataRef,
    OSType             *dataRefType );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II-1131) or `OpenComponent` (II-1130).

sgOut

Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (III-1757).

dataRef

A pointer to the handle in which the data reference is returned. If you do not need the data reference, set this parameter to `NIL`.

dataRefType

A pointer in which the type of the data reference is returned. If you do not need this information, set this parameter to `NIL`.

SGGetOutputMaximumOffset

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The caller is responsible for disposing of the returned handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Working With Sequence Grabber Outputs” (V–2814)

Related Java Methods

quicktime.std.sg.SGOutput.getDataReference()

SGGetOutputMaximumOffset

Returns the maximum offset for data written to the specified sequence grabber output.

```
ComponentResult SGGetOutputMaximumOffset (
    SeqGrabComponent s,
    SGOutput          sgOut,
    wide              *maxOffset );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

sgOut

Identifies the current sequence grabber output. You obtain this identifier by calling `SGNewOutput` (III–1757).

maxOffset

A pointer to the value of the maximum offset for data written to this output. This value is initialized to $(2^{32}-1)$ on systems with a 32-bit file system, and $(2^{64}-1)$ on systems with a 64-bit file system.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Working With Sequence Grabber Outputs” (V–2814)

Related Java Methods

`quicktime.std.sg.SGOutput.getMaximumOffset()`

See Also

For the corresponding set function, see `SGSetOutputMaximumOffset` (III–1799).

SGGetOutputNextOutput

Returns the next sequence grabber output for the specified output.

```
ComponentResult SGGetOutputNextOutput (
    SeqGrabComponent  s,
    SGOutput          sgOut,
    SGOutput          *nextOut );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

sgOut

Identifies the current sequence grabber output. You obtain this identifier by calling `SGNewOutput` (III–1757).

nextOut

A pointer to the next output to be used. If there is no next output, this value is `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Outputs” (V–2814)

Related Java Methods

`quicktime.std.sg.SGOutput.getNextOutput()`

See Also

For the corresponding set function, see `SGSetOutputNextOutput` (III–1800).

SGGetPause

SGGetPause

Determines whether the sequence grabber is paused.

```
ComponentResult SGGetPause (
    SeqGrabComponent s,
    Byte             *paused );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

paused

A pointer to a field that is to receive a constant (see below) that indicates whether the sequence grabber is currently paused.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

paused Constants

`seqGrabUnpause`

The sequence grabber is not paused.

`seqGrabPause`

The sequence grabber is paused; all channels are stopped.

`seqGrabPauseForMenu`

The sequence grabber is paused in order to display a menu; some or all of the channels may be stopped.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.getPause()`

SGGetPreferredPacketSize

Returns the preferred packet size for the sequence grabber component.

```
ComponentResult SGGetPreferredPacketSize (
    SGChannel c,
```



```
long          *preferredPacketSizeInBytes );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

`preferredPacketSizeInBytes`

The preferred packet size in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

`SGGetPreferredPacketSize` was added in QuickTime 2.5 to support video conferencing applications.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

See Also

For the corresponding set function, see `SGSetPreferredPacketSize` (III–1801).

SGGetSettings

Retrieves the current settings of all channels used by the sequence grabber.

```
ComponentResult SGGetSettings (
    SeqGrabComponent s,
    UserData        *ud,
    long            flags );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

ud

A pointer to a space where the sequence grabber returns a pointer to a `UserDataRecord` (IV–2496) structure that contains the configuration information. Your application is responsible for disposing of this structure when it is done with it.

flags

Reserved for Apple. Set this parameter to 0.

SGGetSoundInputDriver

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Working With Sequence Grabber Settings” (V–2812)

Related Java Methods

```
quicktime.std.movies.media.UserData.fromSequenceGrabber(),  
quicktime.std.sg.SequenceGrabber.getSettings()
```

See Also

For the corresponding set function, see SGSetSettings (III–1801).

SGGetSoundInputDriver

Determines the sound input device currently in use by a sound channel component.

```
long SGGetSoundInputDriver (  
    SGChannel    c );
```

c

The connection identifier for the channel for this operation. You get this value from SGNewChannel (III–1753) or SGNewChannelFromComponent (III–1756).

function result A reference to the sound input device. If your sound channel is not using a sound input device, returns NIL.

Discussion

You may want to gain access to the sound input device if you want to change the device’s configuration.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Configuration Functions for Sound Channel Components” (V–2831), “Working With Sound Channels” (V–2821)

Related Java Methods

```
quicktime.std.sg.SGSoundChannel.getInputDriver(),  
quicktime.sound.SPBDDevice.fromSoundChannel()
```

See Also

For the corresponding set function, see `SGSetSoundInputDriver` (III–1802).

SGGetSoundInputParameters

Retrieves various parameters that relate to sound recording.

```
ComponentResult SGGetSoundInputParameters (
    SGChannel    c,
    short        *sampleSize,
    short        *numChannels,
    OSType       *compressionType );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

sampleSize

A pointer to a field to receive the sample size. Set this field to 8 for 8-bit sound or to 16 for 16-bit sound.

numChannels

A pointer to a field to receive the number of sound channels used by the sound sample. Set this field to 1 for monaural sounds or to 2 for stereo sounds.

compressionType

A pointer to a field to receive the format of the sound data (see below).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

compressionType Constants

`'raw'`

Sound samples are uncompressed, in offset-binary format (that is, sample data values range from 0 to 255).

`'MAC3'`

Sound samples have been compressed by the Sound Manager at a ratio of 3:1.

`'MAC6'`

Sound samples have been compressed by the Sound Manager at a ratio of 6:1.

Version Notes

Introduced in QuickTime 3 or earlier.

SGGetSoundInputRate

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Sound Channel Components” (V–2831), “Working With Sound Channels” (V–2821)

Related Java Methods

`quicktime.std.sg.SGSoundChannel.getSoundInputParameters()`

See Also

For the corresponding set function, see `SGSetSoundInputParameters` (III–1803).

SGGetSoundInputRate

Determines the rate at which the sound channel is collecting sound data.

```
Fixed SGGetSoundInputRate (
    SGChannel c );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

function result A fixed-point number that indicates the number of samples your sound channel collects per second.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Sound Channel Components” (V–2831), “Working With Sound Channels” (V–2821)

Related Java Methods

`quicktime.std.sg.SGSoundChannel.getSoundInputRate()`

See Also

For the corresponding set function, see `SGSetSoundInputRate` (III–1804).

SGGetSoundRecordChunkSize

Determines the amount of sound data the sequence grabber component works with at a time.

```
long SGGetSoundRecordChunkSize (
```

```
SGChannel c );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

function result A long integer that specifies the number of seconds of sound data your channel works with at a time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Sound Channel Components” (V–2831), “Working With Sound Channels” (V–2821)

Related Java Methods

`quicktime.std.sg.SGSoundChannel.getRecordChunkSize()`

See Also

For the corresponding set function, see `SGSetSoundRecordChunkSize` (III–1805).

SGGetSrcVideoBounds

Determines the size of the source video boundary rectangle.

```
ComponentResult SGGetSrcVideoBounds (
    SGChannel c,
    Rect *r );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

r

A pointer to a `Rect` (IV–2399) structure that is to receive information about your channel’s source video boundary rectangle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

For video channel components that work with video digitizer components, the source video boundary rectangle corresponds to the video digitizer’s active source rectangle.

SGGetStorageSpaceRemaining

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

Related Java Methods

`quicktime.std.sg.SGVideoChannel.getSrcVideoBounds()`

SGGetStorageSpaceRemaining

Monitors the amount of space remaining for use during a record operation.

```
ComponentResult SGGetStorageSpaceRemaining (
    SeqGrabComponent s,
    unsigned long *bytes );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

bytes

A pointer to a long integer that is to receive a value indicating the amount of space remaining for the current record operation. If you are recording to memory, this value contains information about the amount of memory remaining. If you are recording to a movie file, this value contains information about the amount of storage space available on the device that holds the file.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You can call this function only after you have started a record operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Characteristics” (V–2813)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.getStorageSpaceRemaining()`

SGGetStorageSpaceRemaining64

Provides a 64-bit version of SGGetStorageSpaceRemaining (III–1736).

```
ComponentResult SGGetStorageSpaceRemaining64 (
    SeqGrabComponent s,
    wide             *bytes );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from OpenDefaultComponent (II–1131) or OpenComponent (II–1130).

bytes

A pointer to a wide integer that is to receive a value indicating the amount of space remaining for the current record operation. If you are recording to memory, this value contains information about the amount of memory remaining. If you are recording to a movie file, this value contains information about the amount of storage space available on the device that holds the file.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeComponents.h

See Also

See SGGetStorageSpaceRemaining (III–1736).

SGGetTextReturnToSpaceValue

Indicates whether the text channel component should replace return characters with spaces.

```
ComponentResult SGGetTextReturnToSpaceValue (
    SGChannel c,
    short     *rettospace );
```

c

The connection identifier for the channel for this operation. You get this value from SGNewChannel (III–1753) or SGNewChannelFromComponent (III–1756).

SGGetTimeBase

rettospace

A pointer to a 16-bit integer. On return, this parameter is `TRUE` if the text channel is replacing return characters with spaces, or `FALSE` if the text channel is not replacing return characters with spaces.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

When you capture text from a closed-caption television source, the text is composed of four lines of text of up to 32 characters each, each line separated by a return character. Sometimes it is useful to replace the return characters with spaces. You can call this function to determine whether the text channel component is replacing return characters with spaces.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Text Channel Support” (V–2874)

Related Java Methods

`quicktime.std.sg.SGTextChannel.getReturnToSpaceValue()`

See Also

For the corresponding set function, see `SGSetTextReturnToSpaceValue` (III–1807).

SGGetTimeBase

Retrieves a reference to the time base that is being used by a sequence grabber component.

```
ComponentResult SGGetTimeBase (
    SeqGrabComponent    s,
    TimeBase             *tb );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

tb

A pointer to a time base identifier, such as that returned by `NewTimeBase` (II–1119).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Working With Sequence Grabber Characteristics” (V–2813)

Related Java Methods

quicktime.std.sg.SequenceGrabber.getTimeBase()

S

SGGetTimeRemaining

Obtains an estimate of the amount of recording time that remains for the current record operation.

```
ComponentResult SGGetTimeRemaining (
    SeqGrabComponent s,
    long *ticksLeft );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from OpenDefaultComponent (II–1131) or OpenComponent (II–1130).

ticksLeft

A pointer to a long integer that is to receive a value indicating an estimate of the amount of time remaining for the current record operation. This value is expressed in system **ticks** (sixtieths of a second).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Working With Sequence Grabber Characteristics” (V–2813)

Related Java Methods

quicktime.std.sg.SequenceGrabber.getTimeRemaining()

SGGetUserVideoCompressorList

SGGetUserVideoCompressorList

Returns the video compression formats to be displayed by the specified sequence grabber video channel.

```
ComponentResult SGGetUserVideoCompressorList (
    SGChannel    c,
    Handle       *compressorTypes );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

compressorTypes

A pointer to handle where the list of video compression formats should be returned.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns a copy of the list of video compression formats previously passed to `SGSetUserVideoCompressorList` (III–1810). If no video compression formats have been set, it sets the `compressorTypes` handle to `NIL`. The caller of this routine is responsible for disposing of the returned handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

See Also

For the corresponding set function, see `SGSetUserVideoCompressorList` (III–1810).

SGGetUseScreenBuffer

Determines whether a video channel is allowed to use an offscreen buffer.

```
ComponentResult SGGetUseScreenBuffer (
    SGChannel    c,
    Boolean      *useScreenBuffer );
```

C

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

`useScreenBuffer`

A pointer to a Boolean value. Set this field to `TRUE` if your channel draws directly to the screen. Set it to `FALSE` if your channel can use an offscreen buffer. If your channel cannot work with offscreen buffers, ignore this value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

See Also

For the corresponding set function, see `SGSetUseScreenBuffer` (III–1811).

SGGetVideoBottlenecks

Determines the callback functions that have been assigned to a video channel.

```
ComponentResult SGGetVideoBottlenecks (
    SGChannel      c,
    VideoBottles  *vb );
```

C

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

`vb`

A pointer to a `VideoBottles` (IV–2501) structure. This function sets the fields of that structure to indicate the callback functions that have been assigned to this video channel. You must set the `procCount` field in the `VideoBottles` structure to 9.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

SGGetVideoCompressor

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Video Channel Callback Functions” (V–2823)

See Also

For the corresponding set function, see `SGSetVideoBottlenecks` (III–1811).

SGGetVideoCompressor

Determines a channel’s current image compression parameters.

```
ComponentResult SGGetVideoCompressor (
    SGChannel          c,
    short              *depth,
    CompressorComponent *compressor,
    CodecQ             *spatialQuality,
    CodecQ             *temporalQuality,
    long               *keyFrameRate );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

depth

A pointer to a field that is to receive the depth at which the image is likely to be viewed. Image compressor components may use the depth as an indication of the color or grayscale resolution of the compressed images. Return the depth value currently in use by your channel component. If this parameter is set to `NIL`, the sequence grabber component is not interested in this information.

compressor

A pointer to a field that is to receive an image compressor identifier. Return the identifier that corresponds to the image compressor your channel is using. If this parameter is set to `NIL`, the sequence grabber component is not interested in this information.

spatialQuality

A pointer to a field that is to receive the desired compressed image quality. Return the current quality value. If this parameter is set to `NIL`, the sequence grabber component is not interested in this information.

temporalQuality

A pointer to a field that is to receive the desired temporal quality of the sequence. This parameter governs the level of compression you desire with respect to information between successive frames in the sequence. Return the

current temporal quality value. If this parameter is set to `NIL`, the sequence grabber component is not interested in this information.

`keyFrameRate`

A pointer to a field that is to receive the maximum number of frames allowed between **key frames**. Key frames provide points from which a temporally compressed sequence may be decompressed. This value controls the frequency at which the image compressor places key frames into the compressed sequence. Return the current key frame rate. If this parameter is set to `NIL`, the sequence grabber component is not interested in this information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

spatialQuality and temporalQuality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

Related Java Methods

`quicktime.std.sg.SGVideoChannel.getCompressor()`

See Also

For the corresponding set function, see `SGSetVideoCompressor` (III–1812).

SGGetVideoCompressorType

SGGetVideoCompressorType

Determines the type of image compression that is being applied to a channel's video data.

```
ComponentResult SGGetVideoCompressorType (
    SGChannel      c,
    OSType         *compressorType );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

compressorType

A pointer to a field that is to receive information about the type of image compression to use. Return a value (see below) that corresponds to one of the image-compression types supported by the Image Compression Manager. You should use `GetCodecNameList` (I–386) to retrieve these names, so that your application can take advantage of new compressor types that may be added in the future.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

compressorType Constants

`'rpza'`

Video compressor.

`'jpeg'`

Photo compressor.

`'rle '`

Animation compressor.

`'raw '`

Raw compressor.

`'smc '`

Graphics compressor.

`'cvid'`

Compact video compressor.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

Related Java Methods

`quicktime.std.sg.SGVideoChannel.getCompressorType()`

See Also

For the corresponding set function, see `SGSetVideoCompressorType` (III–1814).

SGGetVideoDigitizerComponent

Determines the video digitizer component that is providing source video to a video channel component.

```
ComponentInstance SGGetVideoDigitizerComponent (
    SGChannel      c );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

function result A component instance that identifies the connection between your video channel component and its video digitizer component. If your video channel component does not use a video digitizer component, set this returned value to `NIL`.

Discussion

This function allows the sequence grabber component to determine the video digitizer component that is providing source video to your video channel component. For example, the sequence grabber component can use this function to obtain access to the video digitizer component so that the grabber component can set the digitizer’s parameters.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

Related Java Methods

`quicktime.std.sg.SGVideoChannel.getDigitizerComponent()`

SGGetVideoRect

See Also

For the corresponding set function, see `SGSetVideoDigitizerComponent` (III–1815). If the sequence grabber component changes any video digitizer component parameters, it notifies your sequence grabber channel component by calling `SGVideoDigitizerChanged` (III–1822).

SGGetVideoRect

Determines the portion of the source video image that is to be captured.

```
ComponentResult SGGetVideoRect (
    SGChannel      c,
    Rect           *r );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

r

A pointer to a `Rect` (IV–2399) structure that is to receive the dimensions of the rectangle that defines the portion of the source video image your component is going to capture.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

Related Java Methods

`quicktime.std.sg.SGVideoChannel.getVideoRect()`

See Also

For the corresponding set function, see `SGSetVideoRect` (III–1816).

SGGrabCompressComplete

Provides the default behavior for your grab-compress-complete function.

```
ComponentResult SGGrabCompressComplete (
```



```

    SGChannel      c,
    Boolean        *done,
    SGCompressInfo *ci,
    TimeRecord     *tr );

```

c

The connection identifier for the channel for this operation. The sequence grabber provides this value to your grab-compress-complete function.

done

A pointer to a Boolean value. This function sets this value to TRUE when it is done; it sets it to FALSE if the operation is incomplete. The sequence grabber provides this pointer to your grab-compress-complete function.

ci

A pointer to an SGCompressInfo (IV-2432) structure. When the operation is complete, the function fills in this structure with information about the compression operation.

tr

A pointer to a TimeRecord (IV-2486) structure. When the operation is complete, the function uses this structure to indicate when the frame was grabbed.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

SGGrabFrame

Provides the default behavior for your grab function.

```

ComponentResult SGGrabFrame (
    SGChannel      c,
    short          bufferNum );

```

c

The reference that identifies the channel for this operation. The sequence grabber component provides this value to your grab function.

bufferNum

Identifies the buffer. The sequence grabber component provides this value to your grab function.

SGGrabFrameComplete

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

SGGrabFrameComplete

Provides the default behavior for your grab-complete function.

```
ComponentResult SGGrabFrameComplete (  
    SGChannel      c,  
    short          bufferNum,  
    Boolean        *done );
```

c

The reference that identifies the channel for this operation. The sequence grabber provides this value to your grab-complete function.

bufferNum

Identifies the buffer. The sequence grabber provides this value to your grab-complete function.

done

A pointer to a Boolean value. The function sets this value to TRUE if the capture is complete, and sets it to FALSE if the capture is incomplete. The sequence grabber provides this pointer to your grab-complete function.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

SGGrabPict

Lets your application obtain a Picture (IV–2331) structure from a sequence grabber component.

```
ComponentResult SGGrabPict (  
    SeqGrabComponent  s,
```

```

PicHandle      *p,
const Rect     *bounds,
short          offscreenDepth,
long           grabPictFlags );

```

S

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

P

A pointer to a field that is to receive a handle to the `Picture` (IV–2331) structure. If the function cannot create the structure, it sets this handle to `NIL`.

bounds

A pointer to the boundary region for the `Picture` (IV–2331) structure. By default, this rectangle lies in the current graphics port. If you set the `grabPictOffScreen` flag in the `grabPictFlags` parameter to 1, the sequence grabber places the structure in an offscreen graphics world. In this case, the rectangle is interpreted in that offscreen world.

offscreenDepth

The pixel depth for the offscreen graphics world. This parameter is typically set to 0, which chooses the best available depth. If you set the `grabPictOffScreen` flag in the `grabPictFlags` parameter to 1, the sequence grabber places the `Picture` (IV–2331) structure in an offscreen graphics world. You specify the pixel depth of this offscreen graphics world with this parameter. If you are displaying the picture, this parameter is ignored.

grabPictFlags

Contains flags (see below) that control the operation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

grabPictFlags Constants

grabPictOffScreen

Instructs the sequence grabber to place the `Picture` (IV–2331) structure in an offscreen graphics world. Set this flag to 1 to use an offscreen graphics world. In this case, you use the `offscreenDepth` parameter to specify the pixel depth in the offscreen buffer. In addition, the rectangle specified by the `bounds` parameter is applied to the offscreen buffer.

grabPictIgnoreClip

Instructs the sequence grabber to ignore any clipping regions you may have defined for the sequence grabber’s channels. Set this flag to 1 to have the sequence grabber ignore these clipping regions.

SGHandleUpdateEvent

grabPictCurrentImage

Set this flag to 1 to provide the fastest possible image capture. Although this flag may fail under certain circumstances, the failure is recoverable; it just will not return a `Picture` (IV–2331) structure. You can then call `SGGrabPict` again without the flag set. This routine does not pause the current **preview** or grab the next frame. It just causes the currently displayed image to be captured. It's a good idea to call `SGPause` (III–1771) before calling `SGGrabPict` with this flag.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

```
quicktime.qd.Pict.fromSequenceGrabber(),  
quicktime.std.sg.SequenceGrabber.grabPict()
```

SGHandleUpdateEvent

Requests that a sequence grabber handle an update event.

```
ComponentResult SGHandleUpdateEvent (  
    SeqGrabComponent    s,  
    const EventRecord    *event,  
    Boolean              *handled );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

event

A pointer to an `EventRecord` (IV–2246) structure.

handled

A pointer to a `Boolean` that returns `TRUE` if the event was handled, `FALSE` otherwise.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

SGIdle

Provides processing time for sequence grabber components.

```
ComponentResult SGIdle (
    SeqGrabComponent s );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

After starting a **preview** or record operation, the application calls this function as often as possible. The sequence grabber component then calls your `SGIdle` function. This continues until the calling application stops the operation by calling `SGStop` (III–1819). `SGIdle` function reports several status and error conditions by means of its result code. If your component returns a nonzero result code during a record operation, the application should call `SGStop` so that the sequence grabber component can store the data it has collected.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Channel Components” (V–2824), “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.idle()`

SGInitChannel

Initializes a channel component.

```
ComponentResult SGInitChannel (
    SGChannel c,
    SeqGrabComponent owner );
```

SGInitialize

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

owner

Identifies the sequence grabber component that has been connected to your channel component. You should save this value so that your channel component can call the utility functions that are provided by the sequence grabber component.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Sequence Grabber Channel Components” (V–2824)

SGInitialize

Initializes the sequence grabber component.

```
ComponentResult SGInitialize (  
    SeqGrabComponent s );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Before you can call this function you must establish a connection to the sequence grabber component. Use `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130) to establish a component connection, as shown below:

```
// SGInitialize coding example  
// See “Discovering QuickTime,” page 262  
SeqGrabComponent MakeMySequenceGrabber (WindowPtr pMacWnd)  
{  
    SeqGrabComponent seqGrab = NIL;
```

```

    OSErr                nErr = noErr;

    // open the default sequence grabber
    seqGrab = OpenDefaultComponent(SeqGrabComponentType, 0);
    if (seqGrab != NIL) {
        // initialize the default sequence grabber component
        nErr = SGInitialize(seqGrab);

        if (nErr == noErr) {
            // set its graphics world to the specified window
            nErr = SGSetGWorld(seqGrab, (CGrafPtr)pMacWnd, NIL);
        }
    }

    if (nErr && (seqGrab != NIL)) {    // clean up on failure
        CloseComponent(seqGrab);
        seqGrab = NIL;
    }
    return seqGrab;
}

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Configuring Sequence Grabber Components” (V-2809)

Related Java Methods

quicktime.std.sg.SequenceGrabber.SequenceGrabber()

SGNewChannel

Creates a sequence grabber channel and assigns a channel component to the channel.

```

ComponentResult SGNewChannel (
    SeqGrabComponent    s,
    OSType               channelType,
    SGChannel            *ref );

```

SGNewChannel

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

channelType

The type of channel to open (see below). This value corresponds to the component subtype value of the channel component.

ref

A pointer to the `frameChannel` field in the `SeqGrabFrameInfo` (IV–2430) structure that is to receive a reference to the channel that is added to the sequence grabber component. If the sequence grabber component successfully locates and connects to an appropriate channel component, the sequence grabber component returns a reference to the channel component into this field.

function result See “Error Codes” (IV–2663). If the sequence grabber component cannot open a connection, it sets the result code to a nonzero value. It returns `noErr` if there is no error.

channelType Constants

`VideoMediaType`

Video channel.

`SoundMediaType`

Sound channel.

Discussion

The channel component is responsible for providing digitized data to the sequence grabber component. You specify the type of channel component to be added to the sequence grabber component, as shown in the following sample code:

```
// SGNewChannel coding example
// See “Discovering QuickTime,” page 263
void MakeMyGrabChannels (SeqGrabComponent    seqGrab,
                        SGChannel             *sgchanVideo,
                        SGChannel             *sgchanSound,
                        const Rect             *rect,
                        Boolean                bWillRecord)
{
    OSErr          nErr;
    long            lUsage;

    // figure out the usage
    lUsage = seqGrabPreview;           // always previewing
```



```

if (bWillRecord)
    lUsage |= seqGrabRecord;           // sometimes recording

// create a video channel
nErr = SGNewChannel(seqGrab, VideoMediaType, sgchanVideo);
if (nErr == noErr) {
    // set boundaries for new video channel
    nErr = SGSetChannelBounds(*sgchanVideo, rect);
    // set usage for new video channel
    if (nErr == noErr)
        nErr = SGSetChannelUsage(*sgchanVideo, lUsage |
                                   seqGrabPlayDuringRecord);

    if (nErr != noErr) {
        // clean up on failure
        SGDisposeChannel(seqGrab, *sgchanVideo);
        *sgchanVideo = NIL;
    }
}

// create a sound channel
nErr = SGNewChannel(seqGrab, SoundMediaType, sgchanSound);
if (nErr == noErr) {
    // set usage of new sound channel
    nErr = SGSetChannelUsage(*sgchanSound, lUsage);
    if (nErr != noErr) {
        // clean up on failure
        SGDisposeChannel(seqGrab, *sgchanSound);
        *sgchanSound = NIL;
    }
}
}

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Configuring Sequence Grabber Components” (V-2809)

Related Java Methods

```

quicktime.std.sg.SGVideoChannel.SGVideoChannel(),
quicktime.std.sg.SGMusicChannel.SGMusicChannel(),
quicktime.std.sg.SGSoundChannel.SGSoundChannel(),

```

SGNewChannelFromComponent

```
quicktime.std.sg.SGTextChannel.SGTextChannel()
```

See Also

If you want to use a specific channel component, you may use `SGNewChannelFromComponent` (III–1756).

SGNewChannelFromComponent

Creates a sequence grabber channel and assigns a channel component to the channel.

```
ComponentResult SGNewChannelFromComponent (  
    SeqGrabComponent    s,  
    SGChannel            *newChannel,  
    Component            sgChannelComponent );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

newChannel

A pointer to a channel component that is to receive a reference to the channel that is added to the sequence grabber component. If the sequence grabber component successfully locates and connects to the specified channel component, the sequence grabber component returns a reference to the channel component into this field.

sgChannelComponent

Identifies the channel component to use. You supply a component ID value to the sequence grabber. The sequence grabber then opens a connection to that channel component and returns your connection ID in the field specified by the `newChannel` parameter. You may obtain a component ID value by calling `FindNextComponent` (I–360).

function result See “Error Codes” (IV–2663). If the sequence grabber component cannot open a connection, it sets the result code to a nonzero value. It returns `noErr` if there is no error.

Discussion

This function is similar to `SGNewChannel` (III–1753), except that this function allows you to specify a particular component rather than just a component subtype value.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Sequence Grabber Components” (V–2809)

See Also

When you are done with the sequence grabber component, you can dispose of the channels you have used by calling `SGDisposeChannel` (III–1695).

SGNewOutput

Creates a new sequence grabber output.

```
ComponentResult SGNewOutput (
    SeqGrabComponent s,
    Handle            dataRef,
    OSType            dataRefType,
    long              whereFlags,
    SGOutput          *sgOut );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

dataRef

A handle to the destination container.

dataRefType

The type of data reference; see “Data References” (IV–2661). If the data reference is an **alias**, you must set the parameter to `rAliasType`.

whereFlags

Flags (see below) that control the record operation. You must set either `seqGrabToDisk` or `seqGrabToMemory` to 1. Set unused flags to 0.

sgOut

A pointer to a sequence grabber output. The sequence grabber component returns an output identifier that you can use with other sequence grabber component functions.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

whereFlags Constants

`seqGrabToDisk`

Instructs the sequence grabber component to write the recorded data to a QuickTime movie in the movie file specified by the `movieFile` parameter. If this

flag is set to 1, the sequence grabber writes the data to the file as the data is recorded.

`seqGrabToMemory`

Instructs the sequence grabber component to store the recorded data in memory until the recording process is complete. The sequence grabber then writes the recorded data to the movie file specified by the `movieFile` parameter. This technique provides better performance than recording directly to the movie file, but it limits the amount of data you can record. If this flag is set to 1, the sequence grabber component is recording to memory.

`seqGrabDontUseTempMemory`

Prevents the sequence grabber component from using temporary memory during the record operation. By default, the sequence grabber component and its channel components use as much temporary memory as necessary to perform the record operation. If this flag is set to 1, the sequence grabber component and its channel components do not use temporary memory.

`seqGrabAppendToFile`

Directs the sequence grabber component to add the recorded data to the data fork of the movie file specified by the `movieFile` parameter. By default, the sequence grabber component deletes the movie file and creates a new file containing one movie and its movie **resource**. If this flag is set to 1, the sequence grabber component appends the recorded data to the data fork of the movie file and creates a new movie resource in that file.

`seqGrabDontAddMovieResource`

Prevents the sequence grabber component from adding the new movie **resource** to the movie file specified by the `movieFile` parameter. By default, the sequence grabber component creates a new movie resource and adds that resource to the movie file. If this flag is set to 1, the sequence grabber component does not add the movie resource to the movie file. You are then responsible for adding the resource to a file, if you so desire

`seqGrabDontMakeMovie`

Prevents the sequence grabber component from creating a movie. By default, the sequence grabber component creates a new movie **resource** and adds the captured data to that movie. If this flag is set to 1, the sequence grabber still calls your data function, but does not write any data to the movie file.

Discussion

Once you have created the sequence grabber output, you can use `SGSetChannelOutput` (III–1778) to assign the output to a sequence grabber channel.

Version Notes

A sequence grabber output ties a sequence grabber channel to a specified data reference for the output of captured data. If you are capturing to a single movie file, you can continue to use `SGSetDataOutput` (III–1785) or `SGSetDataRef` (III–1787) to specify the sequence grabber’s destination. However, if you want to capture movie data into several different files or data references, you must use sequence grabber outputs to do so. Even if you are using outputs, you must still use `SGSetDataOutput` or `SGSetDataRef` to identify where the sequence grabber should create the movie **resource**. You are responsible for creating outputs, assigning them to sequence grabber channels, and disposing of them when you are done.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Outputs” (V–2814)

SGPanelCanRun

Lets a sequence grabber component determine whether a panel component can work with the current sequence grabber channel component.

```
ComponentResult SGPanelCanRun (
    SeqGrabComponent s,
    SGChannel        c );
```

s

Identifies the sequence grabber component’s connection to your panel component. See `SGPanelSetGrabber` (III–1767).

c

The connection identifier for the channel for this operation. The sequence grabber component provides you with a connection to the channel component in question. You must determine whether your panel component can operate with this channel component and its associated channel hardware.

function result If your component can work with the specified channel, return a result code of `noErr`. Otherwise, return an appropriate sequence grabber or sequence grabber channel component result code. See “Error Codes” (IV–2663).

Discussion

Set the `channelFlagHasDependency` flag in the `ComponentDescription` (IV–2212) structure of your sequence grabber panel component to cause the sequence grabber component to call this function. Your component should query the channel

SGPanelEvent

component to determine whether you can operate with it. You may want to use channel component functions to determine the characteristics of the digitization source attached to the channel.

Special Considerations

If your panel component can only support a limited number of connections, you should regulate the number of active connections through this function. Return a nonzero result code to indicate to the sequence grabber that your panel component cannot support the current connection.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing Your Panel Component” (V-2835)

SGPanelEvent

Lets a component receive and process dialog events.

```
ComponentResult SGPanelEvent (
    SeqGrabComponent    s,
    SGChannel            c,
    DialogPtr            d,
    short                itemOffset,
    const EventRecord    *theEvent,
    short                *itemHit,
    Boolean               *handled );
```

s

Identifies the sequence grabber component’s connection to your panel component. See `SGPanelSetGrabber` (III-1767).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III-1753) or `SGNewChannelFromComponent` (III-1756).

d

A dialog pointer identifying the settings dialog box.

itemOffset

The offset to your panel’s first item in the dialog box.

`theEvent`

A pointer to an `EventRecord` (IV–2246) structure. This structure contains information identifying the nature of the event.

`itemHit`

A pointer to a field that is to receive the item number in cases where your component handles the event. The number returned is an absolute, not a relative number, so it must be offset by the `itemOffset` parameter.

`handled`

A pointer to a `Boolean` value. Set this `Boolean` value to `TRUE` if you handle the event; set it to `FALSE` if you do not.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Processing Your Panel’s Events” (V–2836)

SGPanelGetDit1

Lets a sequence grabber component determine the dialog items managed by your panel component.

```
ComponentResult SGPanelGetDit1 (
    SeqGrabComponent s,
    Handle *dit1 );
```

`s`

Identifies the sequence grabber component’s connection to your panel component. See `SGPanelSetGrabber` (III–1767).

`dit1`

A pointer to a handle provided by the sequence grabber component. Your component returns the item list in this handle. Your component should resize this handle as appropriate. The sequence grabber component will dispose of this handle after retrieving the item list, so make sure that the item list is not stored in a **resource**.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

SGPanelGetSettings

Discussion

The sequence grabber uses the information returned by this function to build a sequence grabber settings dialog box for the user. The sequence grabber component will open your **resource** file before calling this function, unless you have instructed the sequence grabber component not to open your resource file by setting the `channelFlagDontOpenResFile` flag in your panel component's `ComponentDescription` (IV–2212) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing Your Panel Component” (V–2835)

SGPanelGetSettings

Retrieves a panel's current settings for a sequence grabber component.

```
ComponentResult SGPanelGetSettings (
    SeqGrabComponent  s,
    SGChannel          c,
    UserData           *ud,
    long               flags );
```

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (III–1767).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

ud

A pointer to a `UserDataRecord` (IV–2496) structure. Your component is responsible for creating a new structure and returning it by means of this pointer. Your component is not responsible for disposing of the structure. These settings may be stored as part of a larger sequence grabber configuration and may be stored for a long period of time. Therefore, you should not store values that may change without your knowledge (such as component ID or connection values). You are free to format the data in user data items any way you desire.

flags

Reserved for future use.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Make sure your component can retrieve the settings information from the user data item when this function is called. You may choose to format the data in such a way that other components can parse it easily, thus allowing your component to operate with other panel components.

Special Considerations

You create new user data items by calling `NewUserData` (II–1124). You may then use other Movie Toolbox functions to manipulate the user data items.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing Your Panel’s Settings” (V–2837)

See Also

For the corresponding set function, see `SGPanelSetSettings` (III–1769).

SGPanelGetTitle

Gets the displayed title of a sequence grabber panel.

```
ComponentResult SGPanelGetTitle (
    SeqGrabComponent s,
    Str255           title );
```

s

Identifies the sequence grabber component’s connection to your panel component. See `SGPanelSetGrabber` (III–1767).

title

A string containing the panel’s title.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

SGPanelInstall

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing Your Panel Component” (V–2835)

SGPanelInstall

Installs added items in a sequence grabber settings dialog box before the dialog box is displayed to the user.

```
ComponentResult SGPanelInstall (
    SeqGrabComponent    s,
    SGChannel            c,
    DialogPtr            d,
    short                itemOffset );
```

s

Identifies the sequence grabber component’s connection to your panel component. See `SGPanelSetGrabber` (III–1767).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

d

A dialog pointer identifying the settings dialog box. Your component may use this value to manage its part of the dialog box.

itemOffset

The offset to your panel’s first item in the dialog box. Because sequence grabber components build your dialog items into a larger dialog box containing other items, this value may be different each time your panel component is installed; do not rely on it being the same.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

A sequence grabber component calls this function just before displaying the dialog box to the user. The sequence grabber provides you with information identifying the channel that your panel is to configure, the dialog box, and the offset of your panel’s items into the dialog box. You may use this opportunity to set default dialog values or to initialize your control values.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing Your Panel Component” (V–2835)

SGPanelItem

Receives and processes mouse clicks in the sequence grabber settings dialog box.

```
ComponentResult SGPanelItem (
    SeqGrabComponent    s,
    SGChannel           c,
    DialogPtr           d,
    short               itemOffset,
    short               itemNum );
```

s

Identifies the sequence grabber component’s connection to your panel component. See `SGPanelSetGrabber` (III–1767).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

d

A dialog pointer identifying the settings dialog box.

itemOffset

The offset to your panel’s first item in the dialog box.

itemNum

The item number of the dialog item selected by the user. The sequence grabber provides an absolute item number. It is your responsibility to adjust this value to account for the offset to your panel’s first item in the dialog box.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

A sequence grabber component calls this function whenever the user clicks an item in the settings dialog box. Your component may then perform whatever processing is appropriate, depending upon the item number.

Version Notes

Introduced in QuickTime 3 or earlier.

SGPanelRemove

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Processing Your Panel’s Events” (V–2836)

See Also

Sequence grabber components use your component’s `SGPanelValidateInput` (III–1770) function to validate the current input settings as a whole.

SGPanelRemove

Removes a panel from the sequence grabber settings dialog box.

```
ComponentResult SGPanelRemove (
    SeqGrabComponent s,
    SGChannel         c,
    DialogPtr         d,
    short             itemOffset );
```

s

Identifies the sequence grabber component’s connection to your panel component. See `SGPanelSetGrabber` (III–1767).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

d

A dialog pointer identifying the settings dialog box.

itemOffset

The offset to your panel’s first item in the dialog box.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

A sequence grabber component calls this function just before removing your items from the settings dialog box. The sequence grabber provides you with information identifying the channel your panel is to configure, the dialog box, and the offset of your panel’s items into the dialog box. You may use this opportunity to save any changes you may have made to the dialog box or to retrieve the contents of text items.

Special Considerations

If the sequence grabber opened your **resource** file, it will still be open when it calls this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing Your Panel Component” (V–2835)

See Also

Sequence grabbers call your `SGPanelInstall` (III–1764) function before displaying the settings dialog box to the user.

SGPanelSetEventFilter

Sets the event filter callback for a sequence grabber panel component.

```
ComponentResult SGPanelSetEventFilter (
    SeqGrabComponent    s,
    SGModalFilterUPP    proc,
    long                refCon );
```

s

Identifies the sequence grabber component’s connection to your panel component. See `SGPanelSetGrabber` (III–1767).

proc

An `SGModalFilterProc` (III–2144) callback.

refCon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Processing Your Panel’s Events” (V–2836)

SGPanelSetGrabber

Identifies a sequence grabber component to a panel component.

```
ComponentResult SGPanelSetGrabber (
```

SGPanelSetResFile

```
SeqGrabComponent    s,  
SeqGrabComponent    sg );
```

s

Identifies the sequence grabber component's connection to your panel component.

sg

Identifies a connection to the sequence grabber component that is using your panel component. Your component may use this connection to call sequence grabber component functions.

function result See "Error Codes" (IV-2663). Returns noErr if there is no error.

Discussion

A sequence grabber component calls this function in order to identify itself to your panel component. Your component can use the provided connection to call sequence grabber functions, either to determine the characteristics of the current capture operation or to alter those characteristics.

Special Considerations

This is typically the first function a sequence grabber component calls after opening your panel component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: "Managing Your Panel Component" (V-2835)

SGPanelSetResFile

Lets the sequence grabber pass a **resource** file's reference number.

```
ComponentResult SGPanelSetResFile (  
    SeqGrabComponent    s,  
    short                resRef );
```

s

Identifies the sequence grabber component's connection to your panel component. See SGPanelSetGrabber (III-1767).

resRef

A reference number that identifies your component's **resource** file. After it closes your resource file, the sequence grabber component calls this function and sets this value to 0.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

A sequence grabber component calls this function to pass you your component's **resource** file reference number. By default, the sequence grabber component opens your component's resource file for you. You can use this reference number to retrieve resources from your resource file. The sequence grabber component also calls this function when it closes your component's resource file. In this case, it sets the resRef parameter to 0. The sequence grabber component may close your resource file at any time; you should not count on any particular calling sequence. If you do not want the sequence grabber component to open your resource file, set the channelFlagDontOpenResFile flag in your panel component's ComponentDescription (IV–2212) structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Managing Your Panel Component” (V–2835)

SGPanelSetSettings

Restores a panel's current settings for a sequence grabber component.

```
ComponentResult SGPanelSetSettings (
    SeqGrabComponent  s,
    SGChannel          c,
    UserData           ud,
    long               flags );
```

s

Identifies the sequence grabber component's connection to your panel component. See SGPanelSetGrabber (III–1767).

c

The connection identifier for the channel for this operation. You get this value from SGNewChannel (III–1753) or SGNewChannelFromComponent (III–1756).

SGPanelValidateInput

ud

Identifies a `UserDataRecord` (IV–2496) structure that contains new settings information for your panel. Your component must not dispose of this structure.

flags

Reserved for future use.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Your component originally creates the settings information when the sequence grabber calls `SGPanelGetSettings` (III–1762). The sequence grabber passes this configuration information back to you in the *ud* parameter to this function. Your component should parse the configuration information and use it to establish your panel’s current settings. Note that your component may not be able to accommodate the original settings. For example, because the settings may have been stored for some time, the hardware environment may not be able to support the values in the settings. You should try to make your new settings match the original settings as closely as possible. If you cannot get close enough, return an appropriate sequence grabber or sequence grabber channel result code.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing Your Panel’s Settings” (V–2837)

See Also

For the corresponding get function, see `SGPanelGetSettings` (III–1762).

SGPanelValidateInput

Validates the contents of the user dialog box for a sequence grabber component.

```
ComponentResult SGPanelValidateInput (
    SeqGrabComponent s,
    Boolean          *ok );
```

s

Identifies the sequence grabber component’s connection to your panel component. See `SGPanelSetGrabber` (III–1767).

ok

A pointer to a Boolean value. Set this value to TRUE if the settings are OK; otherwise, set it to FALSE.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The sequence grabber calls this function when the user clicks the OK button. If the user clicks the Cancel button, the sequence grabber does not call this function. You indicate whether the settings are acceptable by setting the Boolean value referred to by the ok parameter. If you set this value to FALSE, the sequence grabber component ignores the OK button in the dialog box.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Processing Your Panel’s Events” (V–2836)

SGPause

Suspends or restarts a sequence grabber record or **preview** operation.

```
ComponentResult SGPause (
    SeqGrabComponent s,
    Byte             pause );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through SGInitChannel (III–1751).

pause

A constant (see below) that instructs your component to suspend or restart the current operation.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

pause Constants

seqGrabUnpause

Restart the current operation.

seqGrabPause

Pause the current operation.

SGPrepare

Discussion

Your component should not release any system **resources** or temporary memory associated with the current operation. You should be ready to restart the operation immediately.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Channel Components” (V–2824), “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.pause()`

SGPrepare

Instructs a sequence grabber to get ready to begin a **preview** or record operation.

```
ComponentResult SGPrepare (
    SeqGrabComponent s,
    Boolean          prepareForPreview,
    Boolean          prepareForRecord );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

`prepareForPreview`

The sequence grabber component sets this parameter to `TRUE` to prepare for a **preview** operation. The sequence grabber component may set both the `prepareForPreview` and `prepareForRecord` parameters to `TRUE`.

`prepareForRecord`

The sequence grabber component sets this parameter to `TRUE` to prepare for a record operation. The sequence grabber component may set both the `prepareForPreview` and `prepareForRecord` parameters to `TRUE`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If you do not call this function before starting a record or **preview** operation, the sequence grabber component makes these preparations when you start the

operation. You cannot call this function after you start a preview or record operation. If you call this function without subsequently starting a record or preview operation, you should call `SGRelease` (III–1773). This allows the sequence grabber component to release any system **resources** it allocated when you called this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Channel Components” (V–2824), “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.prepare()`

SGRelease

Instructs the sequence grabber to release any system **resources** it allocated when you called `SGPrepare` (III–1772).

```
ComponentResult SGRelease (
    SeqGrabComponent s );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You cannot call this function during a record or **preview** operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Channel Components” (V–2824), “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.release()`

SGSetAdditionalSoundRates

SGSetAdditionalSoundRates

Specifies a list of sound sample rates to be included in the sequence grabber's sound settings dialog box.

```
ComponentResult SGSetAdditionalSoundRates (  
    SGChannel    c,  
    Handle       rates );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

rates

A handle containing a list of unsigned 32-bit fixed-point values. The sequence grabber channel determines the number of sample rates contained in the handle, based on the size of the handle. If any of the requested rates are not supported directly by the available sound capture hardware, sound will be captured at one of the available hardware rates and then converted in software to the requested rate.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The sequence grabber channel makes a copy of the additional rates handle, so your application can immediately dispose of it after making this call.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

See Also

For the corresponding get function, see `SGGetAdditionalSoundRates` (III–1697).

SGSetChannelBounds

Specifies a channel's display boundary rectangle.

```
ComponentResult SGSetChannelBounds (  
    SGChannel    c,  
    const Rect    *bounds );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

bounds

A pointer to a `Rect` (IV–2399) structure that defines your channel’s display boundary rectangle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Channel Characteristics” (V–2815)

Related Java Methods

`quicktime.std.sg.VisualChannel.setBounds()`

See Also

For the corresponding get function, see `SGGetChannelBounds` (III–1700).

SGSetChannelClip

Sets a channel’s clipping region.

```
ComponentResult SGSetChannelClip (
    SGChannel      c,
    RgnHandle      theClip );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

theClip

A handle to the new clipping region. You should make a copy of this region; the application may dispose of the region immediately.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

SGSetChannelDevice

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

Related Java Methods

`quicktime.std.sg.VisualChannel.setClip()`

See Also

For the corresponding get function, see `SGGetChannelClip` (III–1700).

SGSetChannelDevice

Assigns a device to a channel.

```
ComponentResult SGSetChannelDevice (  
    SGChannel      c,  
    StringPtr      name );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

name

A pointer to the device’s name string. This name is contained in the `name` field of the appropriate `SGDeviceName` (IV–2434) structure in the `SGDeviceListRecord` (IV–2433) structure that your channel component returns to the `SGGetChannelDeviceList` (III–1701) function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Managing Channel Devices” (V–2828), “Working With Channel Devices” (V–2818)

SGSetChannelMatrix

Sets a channel’s display transformation matrix.

```
ComponentResult SGSetChannelMatrix (
```

```
SGChannel      c,  
const MatrixRecord *m );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

m

A pointer to a `MatrixRecord` (IV–2304) structure. This parameter is set to `NIL` to select the identity matrix.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

Related Java Methods

`quicktime.std.sg.VisualChannel.setMatrix()`

See Also

For the corresponding get function, see `SGGetChannelMatrix` (III–1703). Other channel component functions may affect this matrix. The `SGSetChannelBounds` (III–1774) function sets the matrix values so that the matrix maps the channel’s output to the channel’s boundary rectangle. `SGSetVideoRect` (III–1816) modifies the matrix so that the specified video rectangle appears in the existing destination rectangle.

SGSetChannelMaxFrames

Limits the number of frames that the sequence grabber will capture from a specified channel.

```
ComponentResult SGSetChannelMaxFrames (  
    SGChannel      c,  
    long           frameCount );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

SGSetChannelOutput

frameCount

The maximum number of frames to capture during the **preview** or record operation. The sequence grabber component sets this parameter to –1 to remove the limit.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

You may use this function only after you have prepared the sequence grabber component for a record operation or during an active record operation.

Special Considerations

Note that sequence grabber components clear this value when you prepare for a record operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

See Also

For the corresponding get function, see SGGetChannelMaxFrames (III–1704).

SGSetChannelOutput

Assigns an output to a channel.

```
ComponentResult SGSetChannelOutput (
    SeqGrabComponent    s,
    SGChannel            c,
    SGOutput             sgOut );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from OpenDefaultComponent (II–1131) or OpenComponent (II–1130).

c

The connection identifier for the channel for this operation. You get this value from SGNewChannel (III–1753) or SGNewChannelFromComponent (III–1756).

`sgOut`

Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (III–1757).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Use this function to assign an output to a channel. One output may be assigned to one or more channels. Note that when you call `SGSetDataRef` (III–1787) or `SGSetDataOutput` (III–1785) the sequence grabber component sets every channel to the specified file or container. If you want to use different outputs, you must use this function to assign the channels appropriately.

Version Notes

A sequence grabber output ties a sequence grabber channel to a specified data reference for the output of captured data. If you are capturing to a single movie file, you can continue to use `SGSetDataOutput` (III–1785) or `SGSetDataRef` (III–1787) to specify the sequence grabber’s destination. However, if you want to capture movie data into several different files or data references, you must use sequence grabber outputs to do so. Even if you are using outputs, you must still use `SGSetDataOutput` or `SGSetDataRef` to identify where the sequence grabber should create the movie **resource**. You are responsible for creating outputs, assigning them to sequence grabber channels, and disposing of them when you are done.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Outputs” (V–2814)

Related Java Methods

`quicktime.std.sg.SGOutput.setChannel()`

SGSetChannelPlayFlags

Adjusts the speed and quality with which the sequence grabber displays data from a channel.

```
ComponentResult SGSetChannelPlayFlags (
    SGChannel    c,
    long         playFlags );
```

`c`

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

SGSetChannelPlayFlags

`playFlags`

A long integer that contains flags (see below) that influence channel playback. A sequence grabber component must use one of these values.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

playFlags Constants

`channelPlayNormal`

Your channel component uses its default playback methodology.

`channelPlayFast`

Your channel component sacrifices playback quality to achieve the specified playback rate.

`channelPlayHighQuality`

Your channel component plays the channel’s data at the highest possible quality. This option sacrifices playback rate for the sake of image quality. It may reduce the amount of processor time available to other programs in the computer. This option should not affect the quality of the recorded data, however.

`channelPlayAllData`

Your channel component tries to play all of the data it captures, even the data that is stored in offscreen buffers. This option is useful when you want to be sure that the user sees as much of the captured data as possible. The sequence grabber component sets this flag to 1 to play all the captured data. The sequence grabber component may combine this flag with any of the other values for the `playFlags` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

Related Java Methods

`quicktime.std.sg.SGChannel.setPlayFlags()`

See Also

For the corresponding get function, see `SGGetChannelPlayFlags` (III–1705).

SGSetChannelRefCon

Sets the value of a reference constant that is passed to your callback functions for channel components.

```
ComponentResult SGSetChannelRefCon (
    SGChannel    c,
    long         refCon );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

refCon

A reference constant value that your component should pass to the callback functions that have been assigned to this channel. Use this parameter to point to a data structure containing any information your callbacks need.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

SGSetChannelSettings

Configures a sequence grabber channel.

```
ComponentResult SGSetChannelSettings (
    SeqGrabComponent  s,
    SGChannel         c,
    UserData          ud,
    long              flags );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

SGSetChannelUsage

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

ud

A `UserDataRecord` (IV–2496) structure that contains the configuration information to be used by the channel component.

flags

Reserved for Apple. Set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Settings” (V–2812)

Related Java Methods

`quicktime.std.sg.SGChannel.setSettings()`

See Also

For the corresponding get function, see `SGGetChannelSettings` (III–1707).

SGSetChannelUsage

Specifies how a channel is to be used by the sequence grabber component.

```
ComponentResult SGSetChannelUsage (
    SGChannel      c,
    long           usage );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

usage

Contains flags (see below) specifying how your channel is to be used. The sequence grabber component may set more than one of these flags to 1. It sets unused flags to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

usage Constants

seqGrabRecord

This flag is set to 1 if your channel is being used for recording.

seqGrabPreview

This flag is set to 1 if your channel is being used for **previewing**.

seqGrabPlayDuringRecord

This flag is set to 1 if your channel plays its captured data during a record operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for All Channel Components” (V–2826), “Working With Channel Characteristics” (V–2815)

Related Java Methods

`quicktime.std.sg.SGChannel.setUsage()`

See Also

For the corresponding get function, see `SGGetChannelUsage` (III–1709).

SGSetChannelVolume

Sets a channel’s sound volume.

```
ComponentResult SGSetChannelVolume (
    SGChannel    c,
    short        volume );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

volume

The volume setting of your channel represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from –1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

SGSetCompressBuffer

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Sound Channel Components” (V–2831), “Working With Channel Characteristics” (V–2815)

Related Java Methods

`quicktime.std.sg.AudioChannel.setVolume()`

See Also

For the corresponding get function, see `SGGetChannelVolume` (III–1710).

SGSetCompressBuffer

Allows the sequence grabber component to direct your component to create a filter buffer for your video channel.

```
ComponentResult SGSetCompressBuffer (
    SGChannel      c,
    short          depth,
    const Rect     *compressSize );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

depth

The pixel depth of the filter buffer. If the sequence grabber sets this parameter to 0, use the depth of the video buffer, which the sequence grabber sets with `SGSetChannelBounds` (III–1774).

compressSize

A pointer to a `Rect` (IV–2399) structure that contains the dimensions of the filter buffer. This buffer should be larger than the destination buffer. To stop filtering the input source video data, the sequence grabber component sets this parameter to `NIL` or it sets the coordinates of this rectangle to 0 (specifying an empty rectangle).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Some video source data may contain unacceptable levels of visual noise or artifacts. One technique for removing this noise is to capture the image and then reduce it in

size. During the size reduction process, the noise can be filtered out. Logically, this buffer sits between the source video buffer and the destination rectangle you set with the `SGSetChannelBounds` (III–1774).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

See Also

For the corresponding get function, see `SGGetCompressBuffer` (III–1711).

SGSetDataOutput

Specifies the movie file and options for a sequence grabber record operation.

```
ComponentResult SGSetDataOutput (
    SeqGrabComponent s,
    const FSSpec      *movieFile,
    long              whereFlags );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

movieFile

A pointer to the `FSSpec` (IV–2262) structure that identifies the movie file for this record operation.

whereFlags

Contains flags (see below) that control the record operation. You must set either `seqGrabToDisk` flag or `seqGrabToMemory` to 1. Set unused flags to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

whereFlags Constants

`seqGrabToDisk`

Instructs the sequence grabber component to write the recorded data to a QuickTime movie in the movie file specified by the `movieFile` parameter. If this flag is set to 1, the sequence grabber writes the data to the file as the data is recorded.

SGSetDataOutput

seqGrabToMemory

Instructs the sequence grabber component to store the recorded data in memory until the recording process is complete. The sequence grabber then writes the recorded data to the movie file specified by the `movieFile` parameter. This technique provides better performance than recording directly to the movie file, but it limits the amount of data you can record. If this flag is set to 1, the sequence grabber component is recording to memory.

seqGrabDontUseTempMemory

Prevents the sequence grabber component from using temporary memory during the record operation. By default, the sequence grabber component and its channel components use as much temporary memory as necessary to perform the record operation. If this flag is set to 1, the sequence grabber component and its channel components do not use temporary memory.

seqGrabAppendToFile

Directs the sequence grabber component to add the recorded data to the data fork of the movie file specified by the `movieFile` parameter. By default, the sequence grabber component deletes the movie file and creates a new file containing one movie and its movie **resource**. If this flag is set to 1, the sequence grabber component appends the recorded data to the data fork of the movie file and creates a new movie resource in that file.

seqGrabDontAddMovieResource

Prevents the sequence grabber component from adding the new movie **resource** to the movie file specified by the `movieFile` parameter. By default, the sequence grabber component creates a new movie resource and adds that resource to the movie file. If this flag is set to 1, the sequence grabber component does not add the movie resource to the movie file. You are then responsible for adding the resource to a file, if you so desire

seqGrabDontMakeMovie

Prevents the sequence grabber component from creating a movie. By default, the sequence grabber component creates a new movie **resource** and adds the captured data to that movie. If this flag is set to 1, the sequence grabber still calls your data function, but does not write any data to the movie file.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Sequence Grabber Components” (V-2809)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.setDataOutput()`

See Also

For the corresponding get function, see `SGGetDataOutput` (III–1711).

SGSetDataProc

Specifies a data function for use by the sequence grabber.

```
ComponentResult SGSetDataProc (
    SeqGrabComponent    s,
    SGDataUPP           proc,
    long                refCon );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

proc

A pointer to your data function. To remove your data function, set this parameter to `NIL`.

refCon

A reference constant. The sequence grabber provides this value to your data callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Sequence Grabber Components” (V–2809)

SGSetDataRef

Specifies the destination data reference for a record operation.

```
ComponentResult SGSetDataRef (
    SeqGrabComponent    s,
    Handle              dataRef,
    OSType              dataRefType,
    long                whereFlags );
```

SGSetDataRef

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

`dataRef`

A handle to the information that identifies the destination container.

`dataRefType`

The type of data reference. If the data reference is an **alias**, you must set the parameter to `rAliasType`.

`whereFlags`

Contains flags (see below) that control the record operation. You must set either `seqGrabToDisk` or `seqGrabToMemory` to 1. Set unused flags to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

whereFlags Constants

`seqGrabToDisk`

Instructs the sequence grabber component to write the recorded data to a QuickTime movie in the movie file specified by the `movieFile` parameter. If this flag is set to 1, the sequence grabber writes the data to the file as the data is recorded.

`seqGrabToMemory`

Instructs the sequence grabber component to store the recorded data in memory until the recording process is complete. The sequence grabber then writes the recorded data to the movie file specified by the `movieFile` parameter. This technique provides better performance than recording directly to the movie file, but it limits the amount of data you can record. If this flag is set to 1, the sequence grabber component is recording to memory.

`seqGrabDontUseTempMemory`

Prevents the sequence grabber component from using temporary memory during the record operation. By default, the sequence grabber component and its channel components use as much temporary memory as necessary to perform the record operation. If this flag is set to 1, the sequence grabber component and its channel components do not use temporary memory.

`seqGrabAppendToFile`

Directs the sequence grabber component to add the recorded data to the data fork of the movie file specified by the `movieFile` parameter. By default, the sequence grabber component deletes the movie file and creates a new file containing one movie and its movie **resource**. If this flag is set to 1, the sequence

grabber component appends the recorded data to the data fork of the movie file and creates a new movie resource in that file.

`seqGrabDontAddMovieResource`

Prevents the sequence grabber component from adding the new movie **resource** to the movie file specified by the `movieFile` parameter. By default, the sequence grabber component creates a new movie resource and adds that resource to the movie file. If this flag is set to 1, the sequence grabber component does not add the movie resource to the movie file. You are then responsible for adding the resource to a file, if you so desire

`seqGrabDontMakeMovie`

Prevents the sequence grabber component from creating a movie. By default, the sequence grabber component creates a new movie **resource** and adds the captured data to that movie. If this flag is set to 1, the sequence grabber still calls your data function, but does not write any data to the movie file.

Discussion

This function allows you to specify the destination for a record operation using a data reference, and to specify other options that govern the operation. This function is similar to `SGSetDataOutput` (III–1785), and provides you with an alternative way to specify the destination.

Special Considerations

If you are performing a **preview** operation, you don’t need to use this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Sequence Grabber Components” (V–2809)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.setDataRef()`

See Also

For the corresponding get function, see `SGGetDataRef` (III–1716).

SGSetFlags

Passes control information about the current operation to the sequence grabber component.

```
ComponentResult SGSetFlags (
    SeqGrabComponent s,
```

SGSetFontName

```
long sgFlags );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

sgFlags

Contains a flag (see below) to tell the sequence grabber if you are performing a controlled grab using a frame-addressable source device.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

sgFlags Constant

`sgFlagControlledGrab`

Set this flag to 1 if you are performing a controlled grab using a frame-addressable source device. The sequence grabber and its channel components optimize their operation for this situation. This flag allows the sequence grabber component to trade off speed and quality.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Characteristics” (V–2813)

See Also

For the corresponding get function, see `SGGetFlags` (III–1718).

SGSetFontName

Sets the name of the font to be used to display text for a text channel component.

```
ComponentResult SGSetFontName (  
    SGChannel    c,  
    StringPtr    pstr );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

pstr

A pointer to a Pascal string containing the name of the font.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

If the specified font is available on the system, the text channel uses the font to display text. If the specified font is not available, the text channel uses the default system font. For more information about fonts, see *Inside Macintosh: Text* (listed in the **bibliography**).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Text Channel Support” (V–2874)

Related Java Methods

`quicktime.std.sg.SGTextChannel.setFontName()`

SGSetFontSize

Sets the font size to be used to display text for a text channel component.

```
ComponentResult SGSetFontSize (
    SGChannel    c,
    short        fontSize );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

fontSize

The point size of the font. This value must be a positive integer.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Text Channel Support” (V–2874)

Related Java Methods

`quicktime.std.sg.SGTextChannel.setFontSize()`

SGSetFrameRate

See Also

For more information about fonts and font sizes, see *Inside Macintosh: Text* (listed in the **bibliography**).

SGSetFrameRate

Specifies a video channel's frame rate for recording.

```
ComponentResult SGSetFrameRate (
    SGChannel      c,
    Fixed          frameRate );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

frameRate

The desired frame rate. If this parameter is set to 0, use your channel's default frame rate. Typically, this corresponds to the fastest rate that your channel can support.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

See Also

For the corresponding get function, see `SGGetFrameRate` (III–1719).

SGSetGWorld

Establishes the graphics port and device for a sequence grabber component.

```
ComponentResult SGSetGWorld (
    SeqGrabComponent  s,
    CGrafPtr          gp,
    GDHandle           gd );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

gp

The destination graphics port, which must be a color graphics port. The sequence grabber component always sets this parameter to a valid value. To use the current graphics port, the parameter is set to `NIL`.

gd

A handle to the destination graphics device. The sequence grabber component always sets this parameter to a valid value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You must call this function if you are working with any channels that collect visual data. If you are working only with data that has no visual representation, you do not need to call this function. The sequence grabber component performs this operation implicitly when you call `SGInitialize` (III–1752) and the component uses your application’s current graphics port. To set it to a specified window, use code such as the following:

```
// SGSetGWorld coding example
// See “Discovering QuickTime,” page 262
SeqGrabComponent MakeMySequenceGrabber (WindowPtr pMacWnd)
{
    SeqGrabComponent    seqGrab = NIL;
    OSErr                nErr = noErr;

    // open the default sequence grabber
    seqGrab = OpenDefaultComponent(SeqGrabComponentType, 0);
    if (seqGrab != NIL) {
        // initialize the default sequence grabber component
        nErr = SGInitialize(seqGrab);

        if (nErr == noErr) {
            // set its graphics world to the specified window
            nErr = SGSetGWorld(seqGrab, (CGrafPtr)pMacWnd, NIL);
        }
    }

    if (nErr && (seqGrab != NIL)) {    // clean up on failure
```

SGSetInstrument

```
        CloseComponent(seqGrab);
        seqGrab = NIL;
    }
    return seqGrab;
}
```

Special Considerations

You cannot call this function during a record or **preview** operation, or after you have prepared the sequence grabber component for a record or preview operation by calling `SGPrepare` (III–1772). The window in which the sequence grabber is to draw video frames as defined by this function must be visible before you call `SGPrepare`; otherwise, the sequence grabber does not display the frames properly.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuring Sequence Grabber Channel Components” (V–2824), “Configuring Sequence Grabber Components” (V–2809)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.setGWorld()`

See Also

For the corresponding get function, see `SGGetGWorld` (III–1719).

SGSetInstrument

Sets a tone description for a music sequence grabber channel.

```
ComponentResult SGSetInstrument (
    SGChannel      c,
    ToneDescription *td );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

td

Pointer to a `ToneDescription` (IV–2487) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Related Java Methods

`quicktime.std.sg.SGMusicChannel.setInstrument()`

See Also

For the corresponding get function, see `SGGetInstrument` (III–1721).

SGSetJustification

Sets the alignment to be used to display text for a text channel component.

```
ComponentResult SGSetJustification (
    SGChannel    c,
    short        just );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

just

A constant (see below) that represents the text alignment.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

just Constants

`teFlushDefault`

Default alignment according to the primary line direction.

`teCenter`

Center for all scripts.

`teFlushRight`

Right alignment for all scripts.

`teFlushLeft`

Left alignment for all scripts.

Discussion

You call this function to specify the alignment to be used for text in a text track. The text channel component justifies text relative to the boundaries of its text box.

Version Notes

Introduced in QuickTime 3 or earlier.

SGSetMaximumRecordTime

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Text Channel Support” (V–2874)

Related Java Methods

`quicktime.std.sg.SGTextChannel.setJustification()`

See Also

For more information on text alignment and the text justification constants, see *Inside Macintosh: Text* (listed in the **bibliography**).

SGSetMaximumRecordTime

Limits the duration of a record operation

```
ComponentResult SGSetMaximumRecordTime (
    SeqGrabComponent s,
    unsigned long    ticks );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

ticks

The maximum duration for the record operation, in system **ticks** (sixtieths of a second). Set this parameter to 0 to remove the time limit from the operation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

By default, there is no time limit on a record operation. If you do not set a limit, a record operation will run until it exhausts the Operating System **resources** or you call `SGStop` (III–1819). Memory and disk space are the two major limiting factors.

Special Considerations

You must call this function before you start a sequence grabber record operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Characteristics” (V–2813)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.setMaximumRecordTime()`

See Also

For the corresponding get function, see `SGGetMaximumRecordTime` (III–1723).

SGSetOutputFlags

Configures an existing sequence grabber output.

```
ComponentResult SGSetOutputFlags (
    SeqGrabComponent  s,
    SGOutput          sgOut,
    long              whereFlags );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

sgOut

Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (III–1757).

whereFlags

Contains flags (see below) that control the record operation. You must set either `seqGrabToDisk` or `seqGrabToMemory` to 1. Set unused flags to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

whereFlags Constants

`seqGrabToDisk`

Instructs the sequence grabber component to write the recorded data to a QuickTime movie in the movie file specified by the `movieFile` parameter. If this flag is set to 1, the sequence grabber writes the data to the file as the data is recorded.

`seqGrabToMemory`

Instructs the sequence grabber component to store the recorded data in memory until the recording process is complete. The sequence grabber then writes the recorded data to the movie file specified by the `movieFile` parameter. This technique provides better performance than recording directly to the movie file, but it limits the amount of data you can record. If this flag is set to 1, the sequence grabber component is recording to memory.

SGSetOutputFlags

`seqGrabDontUseTempMemory`

Prevents the sequence grabber component from using temporary memory during the record operation. By default, the sequence grabber component and its channel components use as much temporary memory as necessary to perform the record operation. If this flag is set to 1, the sequence grabber component and its channel components do not use temporary memory.

`seqGrabAppendToFile`

Directs the sequence grabber component to add the recorded data to the data fork of the movie file specified by the `movieFile` parameter. By default, the sequence grabber component deletes the movie file and creates a new file containing one movie and its movie **resource**. If this flag is set to 1, the sequence grabber component appends the recorded data to the data fork of the movie file and creates a new movie resource in that file.

`seqGrabDontAddMovieResource`

Prevents the sequence grabber component from adding the new movie **resource** to the movie file specified by the `movieFile` parameter. By default, the sequence grabber component creates a new movie resource and adds that resource to the movie file. If this flag is set to 1, the sequence grabber component does not add the movie resource to the movie file. You are then responsible for adding the resource to a file, if you so desire.

`seqGrabDontMakeMovie`

Prevents the sequence grabber component from creating a movie. By default, the sequence grabber component creates a new movie **resource** and adds the captured data to that movie. If this flag is set to 1, the sequence grabber still calls your data function, but does not write any data to the movie file.

Discussion

This function lets you configure an existing sequence grabber output.

Version Notes

A sequence grabber output ties a sequence grabber channel to a specified data reference for the output of captured data. If you are capturing to a single movie file, you can continue to use `SGSetDataOutput` (III–1785) or `SGSetDataRef` (III–1787) to specify the sequence grabber’s destination. However, if you want to capture movie data into several different files or data references, you must use sequence grabber outputs to do so. Even if you are using outputs, you must still use `SGSetDataOutput` or `SGSetDataRef` to identify where the sequence grabber should create the movie **resource**. You are responsible for creating outputs, assigning them to sequence grabber channels, and disposing of them when you are done.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Outputs” (V–2814)

Related Java Methods

`quicktime.std.sg.SGOutput.setOutputFlags()`

SGSetOutputMaximumOffset

Specifies the maximum offset for data written to a specified sequence grabber output.

```
ComponentResult SGSetOutputMaximumOffset (
    SeqGrabComponent s,
    SGOutput sgOut,
    const wide *maxOffset );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

sgOut

Identifies the current sequence grabber output. You obtain this identifier by calling `SGNewOutput` (III–1757).

maxOffset

A pointer to the value of the maximum offset for data written to this output.

function result See “Error Codes” (IV–2663). Returns an error if no more outputs are available. Returns `noErr` if there is no error.

Discussion

If an attempt is made to write data beyond the maximum offset, the sequence grabber switches to the next output specified by `SGSetOutputNextOutput` (III–1800). If no more outputs are available, an end-of-file error is returned and recording ends.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Outputs” (V–2814)

Related Java Methods

`quicktime.std.sg.SGOutput.setMaximumOffset()`

SGSetOutputNextOutput

See Also

For the corresponding get function, see `SGGetOutputMaximumOffset` (III–1728).

SGSetOutputNextOutput

Specifies the order in which sequence grabber outputs should be used.

```
ComponentResult SGSetOutputNextOutput (
    SeqGrabComponent    s,
    SGOutput             sgOut,
    SGOutput             nextOut );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

sgOut

The current output to use. When a new output is created, its `nextOut` is set to `NIL`.

nextOut

The next output to be used. To specify that this is the last output, set this value to `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function should not be called while recording.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Outputs” (V–2814)

Related Java Methods

`quicktime.std.sg.SGOutput.setNextOutput()`

See Also

For the corresponding get function, see `SGGetOutputNextOutput` (III–1729).

SGSetPreferredPacketSize

Sets the preferred packet size for the sequence grabber channel component.

```
ComponentResult SGSetPreferredPacketSize (
    SGChannel      c,
    long           preferredPacketSizeInBytes );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

preferredPacketSizeInBytes

The preferred packet size in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

This function was added in QuickTime 2.5 to support video conferencing applications.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

See Also

For the corresponding get function, see `SGGetPreferredPacketSize` (III–1730).

SGSetSettings

Configures a sequence grabber and its channels.

```
ComponentResult SGSetSettings (
    SeqGrabComponent s,
    UserData         ud,
    long             flags );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

SGSetSoundInputDriver

ud

A `UserDataRecord` (IV–2496) structure that contains the configuration information to be used by the sequence grabber.

flags

Reserved for Apple. Set this parameter to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The sequence grabber disposes of any of its current channels before applying this configuration information. It then opens connections to new channels as appropriate.

Special Considerations

You can restore saved settings by using `NewUserDataFromHandle` (II–1124).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With Sequence Grabber Settings” (V–2812)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.setSettings()`

See Also

You may use the `SGGetIndChannel` (III–1720) function to obtain information about each channel that the sequence grabber is using as a result of applying this new configuration.

SGSetSoundInputDriver

Assigns a sound input device to a sound channel.

```
ComponentResult SGSetSoundInputDriver (
    SGChannel      c,
    ConstStr255Param  driverName );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

`driverName`

The name of the sound input device. This is a Pascal string, and it must correspond to a valid sound input device.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Sound Channel Components” (V–2831), “Working With Sound Channels” (V–2821)

Related Java Methods

`quicktime.std.sg.SGSoundChannel.setInputDriver()`

See Also

For the corresponding get function, see `SGGetSoundInputDriver` (III–1732).

SGSetSoundInputParameters

Sets various parameters that relate to sound recording.

```
ComponentResult SGSetSoundInputParameters (
    SGChannel    c,
    short        sampleSize,
    short        numChannels,
    OSType       compressionType );
```

`c`

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

`sampleSize`

The number of bits in each sound sample. This field is set to 8 for 8-bit sound; it is set to 16 for 16-bit sound.

`numChannels`

Indicates the number of sound channels to be used by the sound sample. This field is set to 1 for monaural sounds; it is set to 2 for stereo sounds.

`compressionType`

A constant (see below) that describes the format of the sound data.

function result See “Error Codes” (IV–2663). If your sound device cannot support a

SGSetSoundInputRate

specified parameter value, return an appropriate Sound Manager result code. Return noErr if there is no error.

compressionType Constants

'raw '

Sound samples uncompressed, in offset-binary format (that is, sample data values range from 0 to 255).

'MAC3'

Sound samples compressed by the Sound Manager at a ratio of 3:1.

'MAC6'

Sound samples compressed by the Sound Manager at a ratio of 6:1.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Sound Channel Components” (V–2831), “Working With Sound Channels” (V–2821)

Related Java Methods

`quicktime.std.sg.SGSoundChannel.setSoundInputParameters()`

See Also

For the corresponding get function, see `SGGetSoundInputParameters` (III–1733).

SGSetSoundInputRate

Sets the rate at which the sound channel obtains its sound data.

```
ComponentResult SGSetSoundInputRate (
    SGChannel    c,
    Fixed        rate );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

rate

The rate at which your sound channel is to acquire data. This parameter specifies the number of samples your sound channel is to generate per second. If your sound channel cannot support the specified rate, use the closest available rate that you can support. If this parameter is set to 0, use your default rate.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Sound Channel Components” (V–2831), “Working With Sound Channels” (V–2821)

Related Java Methods

`quicktime.std.sg.SGSoundChannel.setSoundInputRate()`

See Also

For the corresponding get function, see `SGGetSoundInputRate` (III–1734).

SGSetSoundRecordChunkSize

Controls the amount of sound data in each group of sound samples during a record operation.

```
ComponentResult SGSetSoundRecordChunkSize (
    SGChannel     c,
    long          seconds );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

seconds

The number of seconds of sound data your sound channel component is to work with at a time. This parameter is set to a negative fixed-point number to specify a fraction of a second. For example, to set the duration to half a second, `–0.5` is passed in.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

During record operations, the sequence grabber component and its sound channels work with groups of sound samples, referred to as chunks. By default, each chunk contains two seconds of sound data. Smaller chunks use less memory.

Special Considerations

This function may return a fraction.

SGSetTextBackColor

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Sound Channel Components” (V–2831), “Working With Sound Channels” (V–2821)

Related Java Methods

`quicktime.std.sg.SGSoundChannel.setRecordChunkSize()`

See Also

For the corresponding get function, see `SGGetSoundRecordChunkSize` (III–1734).

SGSetTextBackColor

Sets the background color to be used for the text box.

```
ComponentResult SGSetTextBackColor (
    SGChannel      c,
    RGBColor       *theColor );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

theColor

A pointer to an `RGBColor` (IV–2403) structure that contains the new background color.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You call this function to set the background color of a text track. The text channel component uses the specified color as the background of the text box.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Text Channel Support” (V–2874)

Related Java Methods

`quicktime.std.sg.SGTextChannel.setBackColor()`

SGSetTextForeColor

Sets the color to be used to display text.

```
ComponentResult SGSetTextForeColor (
    SGChannel      c,
    RGBColor       *theColor );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

theColor

A pointer to an `RGBColor` (IV–2403) structure that contains the new text color.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You call this function to set the text color for a text track.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Text Channel Support” (V–2874)

Related Java Methods

`quicktime.std.sg.SGTextChannel.setForeColor()`

SGSetTextReturnToSpaceValue

Determines whether the text channel component should replace return characters with spaces.

```
ComponentResult SGSetTextReturnToSpaceValue (
    SGChannel      c,
    short          rettospace );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

rettospace

Set this parameter to `TRUE` if the text channel should replace return characters with spaces, or `FALSE` otherwise.

SGSettingsDialog

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

When you capture text from a closed-caption television source, the text is composed of four lines of text of up to 32 characters each, each line separated by a return character. You can call this function to request that the text channel component replace the return characters with spaces.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Text Channel Support” (V–2874)

Related Java Methods

quicktime.std.sg.SGTextChannel.setReturnToSpaceValue()

See Also

For the corresponding get function, see SGGetTextReturnToSpaceValue (III–1737).

SGSettingsDialog

Causes a sequence grabber to display its settings dialog box to the user.

```
ComponentResult SGSettingsDialog (
    SeqGrabComponent      s,
    SGChannel              c,
    short                  numPanels,
    ConstComponentListPtr  panelList,
    long                   flags,
    SGModalFilterUPP       proc,
    long                   procRefNum );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from OpenDefaultComponent (II–1131) or OpenComponent (II–1130).

c

The connection identifier for the channel for this operation. You get this value from SGNewChannel (III–1753) or SGNewChannelFromComponent (III–1756).

numPanels

The number of panel components to be listed in the panel component pop-up menu. You specify the panel components with the panelList parameter. You

may use these parameters to limit the user's choice of panel components. If you set this parameter to 0 and the `panelList` parameter to `NIL`, the sequence grabber lists all available panel components.

`panelList`

A pointer to an array of component identifiers. The sequence grabber presents only these components in the panel component pop-up menu. You specify the number of identifiers in the array with the `numPanels` parameter. If you set this parameter to `NIL`, the sequence grabber lists all available panel components.

`flags`

Either set this to 0 or to `seqGrabSettingsPreviewOnly` (see below).

`proc`

Specifies an `SGModalFilterProc` (III-2144) callback. Because the sequence grabber's settings dialog box is a movable modal dialog box, you must supply an event filter function to process update events in your window.

`procRefNum`

A reference constant to be passed to your filter callback. Use this parameter to point to a data structure containing any information your function needs.

function result See "Error Codes" (IV-2663). If the user clicks OK and the settings are acceptable to the panel and channel components, this function returns a result code of `noErr`.

flags Constant

`seqGrabSettingsPreviewOnly`

Set this flag to indicate that the user will be using the dialog box provided by this function to configure the sequence grabber for **previewing** only, not for recording. This function automatically excludes any panels that aren't necessary for preview configuring, such as video or audio compression settings. Use this flag for applications that allow a live video signal to be viewed but not captured.

Discussion

Because the user may change several channel configuration parameters, your application should retrieve new configuration information from the channel so that you can update any values you save, such as the channel's display boundaries or the channel device. In particular, the video rectangle for the channels may have to be adjusted.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: "Working With Sequence Grabber Settings" (V-2812)

SGSetUserVideoCompressorList

Related Java Methods

`quicktime.std.sg.SGChannel.settingsDialog()`

SGSetUserVideoCompressorList

Specifies the list of video compression formats to be included in the sequence grabber's video settings dialog box.

```
ComponentResult SGSetUserVideoCompressorList (
    SGChannel      c,
    Handle          compressorTypes );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

compressorTypes

A handle containing a list of `OSType` values indicating which video compression formats should be displayed. See “Codec Identifiers” (IV–2655). The sequence grabber channel determines the number of video compression formats contained in the handle based on the size of the handle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function lets an application limit the number of video compression formats that will be displayed to the user. For applications using the sequence grabber for a very specific purpose, this function allows inappropriate compression choices to be filtered out.

Special Considerations

The sequence grabber channel makes a copy of the video compression formats handle. Therefore, your application can immediately dispose of the video compression formats handle after making this call.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

See Also

For the corresponding get function, see `SGGetUserVideoCompressorList` (III–1740).

SGSetUseScreenBuffer

Controls whether a video channel uses an offscreen buffer.

```
ComponentResult SGSetUseScreenBuffer (
    SGChannel    c,
    Boolean      useScreenBuffer );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

useScreenBuffer

Indicates whether to use an offscreen buffer. If this parameter is set to `TRUE`, draw directly to the screen. If it is set to `FALSE`, your channel may use an offscreen buffer. If your channel cannot work with offscreen buffers, ignore this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Directing a channel to draw offscreen may be useful if you are performing transformations on the data before displaying it (such as blending it with another graphical image).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

See Also

For the corresponding get function, see `SGGetUseScreenBuffer` (III–1740).

SGSetVideoBottlenecks

Assigns callback functions to a video channel.

```
ComponentResult SGSetVideoBottlenecks (
    SGChannel    c,
    VideoBottles *vb );
```

SGSetVideoCompressor

c

The connection identifier for the video channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

vb

A pointer to a `VideoBottles` (IV–2501) structure, which identifies the callback functions to be assigned to the video channel.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Video Channel Callback Functions” (V–2823)

See Also

For the corresponding get function, see `SGGetVideoBottleNecks` (III–1741).

SGSetVideoCompressor

Specifies many of the parameters that control image compression of the video data captured by a video channel.

```
ComponentResult SGSetVideoCompressor (
    SGChannel          c,
    short              depth,
    CompressorComponent compressor,
    CodecQ             spatialQuality,
    CodecQ             temporalQuality,
    long               keyFrameRate );
```

c

The connection identifier for the video channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

depth

The depth at which the image is likely to be viewed. Image compressors may use this as an indication of the color or grayscale resolution of the compressed images. If the sequence grabber component sets this parameter to 0, let the sequence grabber component determine the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 indicate 1-bit, 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your component can determine

which depths are supported by a given compressor by examining the `CodecInfo` (IV-2202) structure returned by `GetCodecInfo` (I-385).

`compressor`

The image compressor identifier. The sequence grabber component may specify a particular compressor by setting this parameter to its compressor identifier. You can obtain these identifiers from `GetCodecNameList` (I-386).

`spatialQuality`

A constant (see below) that defines the desired quality of the compressed image.

`temporalQuality`

A constant (see below) that defines the desired temporal quality of the sequence. This parameter governs the level of compression the sequence grabber component desires with respect to information in successive frames in the sequence. The sequence grabber component sets this parameter to 0 to prevent the image compressor from applying temporal compression to the sequence.

`keyFrameRate`

The maximum number of frames allowed between **key frames**. Key frames provide points from which a temporally compressed sequence may be decompressed. The sequence grabber component uses this parameter to control the frequency with which the image compressor places key frames into the compressed sequence.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

spatialQuality and temporalQuality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

SGSetVideoCompressorType

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

Related Java Methods

`quicktime.std.sg.SGVideoChannel.setCompressor()`

See Also

For the corresponding get function, see `SGGetVideoCompressor` (III–1742).

SGSetVideoCompressorType

Specifies the type of image compression to be applied to captured video images.

```
ComponentResult SGSetVideoCompressorType (
    SGChannel      c,
    OSType         compressorType );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

compressorType

A constant (see below) that defines the type of image compression to use. You should use `GetCodecNameList` (I–386) to retrieve their names, so that your application can take advantage of new compressor types that may be added in the future.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

compressorType Constants

`'rpza'`

Video compressor.

`'jpeg'`

Photo compressor.

'rle '
Animation compressor.

'raw '
Raw compressor.

'smc '
Graphics compressor.

'cvid'
Compact video compressor.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

Related Java Methods

`quicktime.std.sg.SGVideoChannel.setCompressorType()`

See Also

For the corresponding get function, see `SGGetVideoCompressorType` (III–1744).

SGSetVideoDigitizerComponent

Assigns a video digitizer component to a video channel.

```
ComponentResult SGSetVideoDigitizerComponent (
    SGChannel      c,
    ComponentInstance vdig );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

vdig

A component instance that identifies a connection to a video digitizer component. Your video channel component should use this video digitizer component to obtain its source video data.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

SGSetVideoRect

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

Related Java Methods

`quicktime.std.sg.SGVideoChannel.setDigitizerComponent()`

See Also

For the corresponding get function, see `SGGetVideoDigitizerComponent` (III–1745).

SGSetVideoRect

Specifies a part of the source video image that is to be captured by a sequence grabber component.

```
ComponentResult SGSetVideoRect (
    SGChannel      c,
    const Rect     *r );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

r

A pointer to the `Rect` (IV–2399) structure that defines the portion of the source video image to be captured. This rectangle must lie within the boundaries of the source video boundary rectangle, which the sequence grabber can obtain by calling `SGGetSrcVideoBounds` (III–1735). If you do not use this function to set a source rectangle, the sequence grabber component captures the entire video image, as defined by the source video boundary rectangle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You cannot call this function during a record operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

Related Java Methods

`quicktime.std.sg.SGVideoChannel.setVideoRect()`

See Also

For the corresponding get function, see `SGGetVideoRect` (III–1746).

SGSortDeviceList

Sorts a device list alphabetically.

```
ComponentResult SGGSortDeviceList (
    SeqGrabComponent s,
    SGDeviceList list );
```

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` (II–1131) or `OpenComponent` (II–1130).

list

A handle to an `SGDeviceListRecord` (IV–2433) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

SGSoundInputDriverChanged

Notifies the sequence grabber component whenever you change the configuration of a sound channel’s sound input device.

```
ComponentResult SGGSoundInputDriverChanged (
    SGChannel c );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

SGStartPreview

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Configuration Functions for Sound Channel Components” (V–2831), “Working With Sound Channels” (V–2821)

Related Java Methods

quicktime.std.sg.SGSoundChannel.setInputDriver()

SGStartPreview

Instructs the sequence grabber to begin processing data from its channels.

```
ComponentResult SGStartPreview (  
    SeqGrabComponent s );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through SGInitChannel (III–1751).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Your channel component should immediately present the data to the user in the appropriate format, according to your channel’s configuration. Display video data in the destination display region; play sound data at the specified volume settings.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Controlling Sequence Grabber Channel Components” (V–2824), “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

quicktime.std.sg.SequenceGrabber.startPreview()

See Also

In **preview** mode, the sequence grabber does not save any of the data it gathers from its channels. If you want to record the data, use record mode. You start a record operation by calling `SGStartRecord` (III–1819).

SGStartRecord

Instructs the sequence grabber component to begin collecting data from its channels.

```
ComponentResult SGStartRecord (
    SeqGrabComponent s );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Channel Components” (V–2824), “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

```
quicktime.std.sg.SequenceGrabber.startRecord()
```

See Also

You can cause the sequence grabber to display the data it obtains from its channels without storing any of the data by calling `SGStartPreview` (III–1818).

SGStop

Stops a **preview** or record operation.

```
ComponentResult SGStop (
    SeqGrabComponent s );
```

SGTransferFrameForCompress

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

It is dangerous to allow an update event to occur during recording. Many digitizers capture directly to the screen, and an update event will result in data loss.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Channel Components” (V–2824), “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.stop()`

SGTransferFrameForCompress

Provides the default behavior for your transfer-frame function.

```
ComponentResult SGTransferFrameForCompress (
    SGChannel      c,
    short          bufferNum,
    const MatrixRecord *mp,
    RgnHandle      clipRgn );
```

c

The reference that identifies the channel for this operation. The sequence grabber component provides this value to your transfer-frame function.

bufferNum

Identifies the buffer. The sequence grabber component provides this value to your transfer-frame function.

mp

A pointer to a `MatrixRecord` (IV–2304) structure for the transfer operation. If there is no matrix for the operation, set this parameter to `NIL`.

`clipRgn`

A handle to a `MacRegion` (IV–2303) structure that defines the clipping region for the destination image. This region is defined in the destination coordinate system. If there is no clipping region, set this parameter to `NIL`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

S

SGUpdate

Notifies your component about update events, to update its display.

```
ComponentResult SGUpdate (
    SeqGrabComponent s,
    RgnHandle         updateRgn );
```

`s`

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

`updateRgn`

Indicates the part of the window that has been changed.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Applications can determine the part of the window that has been changed by examining the appropriate window record. For example, they may call the sequence grabber in this manner:

```
SGUpdate (theSG, ((WindowPeek)updateWindow)->updateRgn);
```

Special Considerations

Your application should avoid drawing where the sequence grabber is displaying video. Doing so may cause some video digitizer components to stop displaying video.

Version Notes

Introduced in QuickTime 3 or earlier.

SGVideoDigitizerChanged

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Channel Components” (V–2824), “Controlling Sequence Grabber Components” (V–2811)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.update()`

SGVideoDigitizerChanged

Notifies the sequence grabber component whenever you change the configuration of a video channel’s video digitizer.

```
ComponentResult SGVideoDigitizerChanged (
    SGChannel    c );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

It is very important to notify the sequence grabber of any configuration changes you make.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Configuration Functions for Video Channel Components” (V–2828), “Working With Video Channels” (V–2819)

Related Java Methods

`quicktime.std.sg.SGVideoChannel.digitizerChanged()`

SGWriteExtendedMovieData

Allows your channel component to add data to a movie.

```
ComponentResult SGWriteExtendedMovieData (
    SeqGrabComponent  s,
    SGChannel          c,
```

```
Ptr          p,
long         len,
wide        *offset,
SGOutput     *sgOut );
```

S

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

C

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

p

The location of the data to be added to the movie.

len

The number of bytes of data to be added to the movie.

offset

A pointer to a wide integer that receives the offset to the new data in the movie. If the movie is in memory, the returned offset reflects the location the data will have in the movie on a permanent storage device.

sgOut

A pointer to the sequence grabber output to which the data was written.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function differs from `SGWriteMovieData` (III–1824), in two respects: the `offset` parameter has a 64-bit value, and the `sgOut` parameter does not exist in `SGWriteMovieData`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

Related Java Methods

`quicktime.std.sg.SequenceGrabber.writeExtendedMovieData()`

See Also

See `SGWriteMovieData` (III–1824).

SGWriteMovieData

SGWriteMovieData

Lets a channel component add data to a movie.

```
ComponentResult SGWriteMovieData (  
    SeqGrabComponent    s,  
    SGChannel            c,  
    Ptr                  p,  
    long                 len,  
    long                 *offset );
```

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (III–1751).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

p

The location of the data to be added to the movie.

len

The number of bytes of data to be added to the movie.

offset

A pointer to a long integer that is to receive the offset to the new data in the movie. The sequence grabber component returns an offset that is correct in the context of a movie **resource**, even if the movie data is currently stored in memory. That is, if the movie is in memory, the returned offset reflects the location that the data will have in a movie on a permanent storage device, such as a disk.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Utility Functions for Sequence Grabber Channel Components” (V–2833)

See Also

See `SGWriteExtendedMovieData` (III–1822).

SGWriteSamples

Called by a sequence grabber component when it is ready to add recorded data to a movie.

```
ComponentResult SGWriteSamples (
    SGChannel      c,
    Movie          m,
    AliasHandle     theFile );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

m

Identifies the movie to which your component should add the captured data. Your component should not make any other changes to the movie identified by this reference. Use `SGWriteMovieData` (III–1824) instead.

theFile

Identifies the movie file. The sequence grabber component provides this **alias** so that you can supply it to the Movie Toolbox. You should not open this file or write to it directly. Use `SGWriteMovieData` (III–1824) instead.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Sequence Grabber Channel Components” (V–2824)

ShowHideTaskBar

Shows or hides the Windows taskbar.

```
void ShowHideTaskBar (
    Boolean    showIt );
```

showIt

Pass `TRUE` to show the task bar, `FALSE` otherwise.

ShowMovieInformation

Discussion

This call can be used to hide the taskbar during full-screen movie playback.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

ShowMovieInformation

Displays a movie's information.

```
void ShowMovieInformation (
    Movie          theMovie,
    ModalFilterUPP filterProc,
    long           refCon );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

filterProc

A **Universal Procedure Pointer** that accesses a `ModalFilterProc` (III–2098) callback.

refCon

A reference constant to be passed to your filter callback. Use this parameter to point to a data structure containing any information your function needs.

function result

You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Related Java Methods

`quicktime.std.movies.Movie.showInformation()`

ShowMoviePoster

Displays a movie's **poster**.

```
void ShowMoviePoster (
    Movie    theMovie );
```

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

The Movie Toolbox draws the movie **poster** once, in the movie's graphics world, using the movie's matrix and display clipping characteristics. You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Special Considerations

This function works on both active and inactive movies.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movie Posters and Movie Previews” (V–2718)

Related Java Methods

```
quicktime.std.movies.Movie.showPoster()
```

SkewMatrix

Modifies the contents of a matrix so that it defines a skew transformation.

```
void SkewMatrix (
    MatrixRecord    *m,
    Fixed           skewX,
    Fixed           skewY,
    Fixed           aboutX,
    Fixed           aboutY );
```

SndAddModifier

m

A pointer to the matrix for this operation. The `SkewMatrix` function updates the contents of the `MatrixRecord` (IV-2304) structure so that it defines a skew operation; it concatenates the respective transformations onto whatever was initially in the matrix structure. You specify the magnitude and direction of the skew operation with the `skewX` and `skewY` parameters. You specify an anchor point with the `aboutX` and `aboutY` parameters.

`skewX`

The skew value to be applied to x coordinates.

`skewY`

The skew value to be applied to y coordinates.

`aboutX`

The x coordinate of the anchor point.

`aboutY`

The y coordinate of the anchor point.

Discussion

A skew operation alters the display of an element along one dimension. For example, converting a rectangle into a parallelogram is a skew operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Managing Matrices” (V-2764)

Related Java Methods

`quicktime.std.image.Matrix.skew()`

SndAddModifier

Obsolete.

```
OSErr SndAddModifier (
    SndChannelPtr  chan,
    Ptr            modifier,
    short          id,
    long           init );
```

Version Notes

Introduced in QuickTime 3 or earlier. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: Sound.h

SndChannelStatus

Determines the status of a sound channel.

```
OSErr SndChannelStatus (
    SndChannelPtr    chan,
    short            theLength,
    SCStatusPtr       theStatus );
```

chan

A pointer to a valid sound channel.

theLength

The size in bytes of the sound channel status record. You should set this field to `SizeOf(SCStatus)`.

theStatus

A pointer to an `SCStatus` (IV–2425) structure. On return, the fields of this structure accurately describe the sound channel specified by the *chan* parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SndControl

Obsolete.

```
OSErr SndControl (
    short    id,
```

SndDisposeChannel

```
SndCommand    *cmd );
```

Version Notes

Introduced in QuickTime 3 or earlier. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: Sound.h

SndDisposeChannel

Disposes of the queue of sound commands associated with a sound channel.

```
OSErr SndDisposeChannel (
    SndChannelPtr    chan,
    Boolean           quietNow );
```

chan

A pointer to a valid `SndChannel` (IV-2440) structure.

quietNow

A `Boolean` value that indicates whether the channel should be disposed immediately (`TRUE`) or after sound stops playing (`FALSE`). This function can dispose of a channel immediately or wait until the queued commands are processed. If `quietNow` is set to `TRUE`, a `flushCmd` command and then a `quietCmd` command are sent to the channel, bypassing the command queue. This removes all commands, stops any sound in progress, and closes the channel. If `quietNow` is set to `FALSE`, then the Sound Manager issues a `quietCmd` command only; it does not bypass the command queue, and it waits until the `quietCmd` command is processed before disposing of the channel.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

If your application created its own sound channel record in memory or installed a sound as a voice in a channel, the Sound Manager does not dispose of that memory. The Sound Manager also does not release memory associated with a sound **resource** that you have played on a channel. You might use the `userInfo` field of the `SndChannel` (IV-2440) structure to store the address of a sound handle you wish to release before disposing of the sound channel itself.

Special Considerations

Because this function might dispose of memory, you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SndDoCommand

Queues a command in a sound channel.

```
OSErr SndDoCommand (
    SndChannelPtr      chan,
    const SndCommand   *cmd,
    Boolean             noWait );
```

chan

A pointer to a valid `SndChannel` (IV–2440) structure.

cmd

A `SndCommand` (IV–2441) structure to be sent to the channel specified in the *chan* parameter.

noWait

A flag indicating whether the Sound Manager should wait for a free space in a full queue (FALSE) or whether it should return immediately with a `queueFull` result code if the queue is full (TRUE). This parameter has meaning only if a sound channel's queue of sound commands is full. If the *noWait* parameter is set to FALSE and the queue is full, the Sound Manager waits until there is space to add the command, thus preventing your application from doing other processing. If *noWait* is set to TRUE and the queue is full, the Sound Manager does not send the command and returns the `queueFull` result code.

function result See “Error Codes” (IV–2663). If *noWait* is set to TRUE and the queue is full, the Sound Manager does not send the command and returns the `queueFull` result code. Returns `noErr` if there is no error.

Special Considerations

Whether this function moves memory depends on the particular sound command you're sending it. Most of the available sound commands do not cause it to move memory and can therefore be issued at interrupt time. Moreover, you can sometimes safely send commands at interrupt time that would otherwise cause memory to move if you've previously issued the `soundCmd` sound command to preconfigure the channel at noninterrupt time.

SndDoImmediate

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

See “Sound Commands” (IV–2691).

SndDoImmediate

Places a sound command in front of a sound channel’s command queue.

```
OSErr SndDoImmediate (
    SndChannelPtr      chan,
    const SndCommand   *cmd );
```

chan

A pointer to a valid `SndChannel` (IV–2440) structure.

cmd

A `SndCommand` (IV–2441) structure to be sent to the channel specified in the *chan* parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function operates much like `SndDoCommand` (III–1831), except that it bypasses the existing command queue of the sound channel and sends the specified command directly to the Sound Manager for immediate processing. This function also overrides any `waitCmd`, `pauseCmd`, or `syncCmd` commands that might have already been processed. However, other commands already received by the Sound Manager will not be interrupted by this function, although a `quietCmd` command sent via `SndDoImmediate` (III–1832) will quiet a sound already playing.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

See “Sound Commands” (IV–2691).

SndGetInfo

Retrieves information about an open Sound Manager channel, including hardware settings.

```
OSErr SndGetInfo (
    SndChannelPtr    chan,
    OSType           selector,
    void             *infoPtr );
```

chan

A pointer to a valid `SndChannel` (IV–2440) structure.

selector

A sound input device information selector that specifies the type of information you need. See “Sound Information Selectors” (IV–2693).

infoPtr

A pointer to a buffer in which information should be returned. This buffer must be large enough to contain the type of information specified in the *selector* parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function should be used instead of attempting to communicate directly with sound components.

Special Considerations

This function uses a selector-based interface similar to `SPBGetDeviceInfo` (III–1871), with the same sound information selectors.

Version Notes

Available only with Sound Manager version 3.1 or later. Check for this by calling `SndSoundManagerVersion` (III–1847) to get the installed version number.

Programming Info

C interface file: `Sound.h`

See Also

See `SPBGetDeviceInfo` (III–1871). For the corresponding set function, see `SndSetInfo` (III–1845).

SndGetSysBeepState

Determines if the system alert sound is enabled.

SndInputGetDeviceInfo

```
void SndGetSysBeepState (
    short    *sysBeepState );
```

sysBeepState

On return, a pointer to one of two constants (see below) that represents the state of the system alert sound.

sysBeepState Constants

sysBeepDisable

The system alert sound is disabled.

sysBeepEnable

The system alert sound is enabled.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

See Also

For the corresponding set function, see SndSetSysBeepState (III–1846).

SndInputGetDeviceInfo

Undocumented

```
ComponentResult SndInputGetDeviceInfo (
    ComponentInstance    self,
    OSType                infoType,
    void                  *infoData );
```

self

Undocumented

infoType

A sound input device information selector that specifies the type of information you need. See “Sound Information Selectors” (IV–2693).

infoData

A pointer to a buffer in which information should be returned. This buffer must be large enough to contain the type of information specified in the infoType parameter.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

See Also

For the corresponding set function, see SndInputSetDeviceInfo (III–1838).

SndInputGetStatus

Undocumented

```
ComponentResult SndInputGetStatus (
    ComponentInstance    self,
    short                *recordingStatus,
    unsigned long         *totalSamplesToRecord,
    unsigned long         *numberOfSamplesRecorded );
```

self

Undocumented

recordingStatus

Undocumented

totalSamplesToRecord

Undocumented

numberOfSamplesRecorded

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SndInputInitHardware

Undocumented

```
ComponentResult SndInputInitHardware (
```

SndInputPauseRecording

```
ComponentInstance self );

self
    Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SndInputPauseRecording

```
Undocumented

ComponentResult SndInputPauseRecording (
    ComponentInstance self );

self
    Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SndInputReadAsync

```
Undocumented

ComponentResult SndInputReadAsync (
    ComponentInstance self,
    SndInputCmpParamPtr SICParamPtr );

self
    Undocumented

SICParamPtr
    A pointer to a SndInputCmpParam (IV–2446) structure.
```

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SndInputReadSync

Undocumented

```
ComponentResult SndInputReadSync (
    ComponentInstance    self,
    SndInputCmpParamPtr  SICParmPtr );
```

self

Undocumented

SICParmPtr

A pointer to a SndInputCmpParam (IV–2446) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SndInputResumeRecording

Undocumented

```
ComponentResult SndInputResumeRecording (
    ComponentInstance    self );
```

self

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

SndInputSetDeviceInfo

Programming Info

C interface file: Sound.h

SndInputSetDeviceInfo

Undocumented

```
ComponentResult SndInputSetDeviceInfo (  
    ComponentInstance    self,  
    OSType               infoType,  
    void                 *infoData );
```

self

Undocumented

infoType

A sound input device information selector that specifies the type of information you need. See “Sound Information Selectors” (IV–2693).

infoData

A pointer to a buffer in which information should be returned. This buffer must be large enough to contain the type of information specified in the infoType parameter.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

See Also

For the corresponding get function, see SndInputGetDeviceInfo (III–1834).

SndInputStopRecording

Undocumented

```
ComponentResult SndInputStopRecording (  
    ComponentInstance    self );
```

self

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SndManagerStatus

Determines information about all sound channels currently allocated.

```
OSErr SndManagerStatus (
    short          theLength,
    SMStatusPtr    theStatus );
```

theLength

The size in bytes of the SMStatus (IV–2438) structure.

theStatus

A pointer to an SMStatus (IV–2438) structure.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

If this function executes successfully, the fields of the structure specified by *theStatus* accurately describe the current status of the Sound Manager.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SndNewChannel

Allocates a new sound channel.

```
OSErr SndNewChannel (
    SndChannelPtr    *chan,
    short            synth,
    long              init,
    SndCallbackUPP    userRoutine );
```

SndNewChannel

`chan`

A pointer to a valid `SndChannel` (IV–2440) structure. You can pass a pointer whose value is `NIL` to force the Sound Manager to allocate the `SndChannel` structure internally. If you do this, the function allocates the structure in your application’s heap and returns a pointer to it. If you pass a pointer to `NIL` in this parameter, the function allocates enough memory to store 128 sound commands. However, if you pass a pointer to a `SndChannel` structure rather than a pointer to `NIL`, the amount of memory allocated is determined by the `qLength` field of that structure. Thus, if you wish to control the size of the sound queue, you must allocate your own `SndChannel` structure.

`synth`

A constant (see below) that identifies the sound data type you intend to play on this channel. If you do not want to specify a specific data type, pass 0 in this parameter. You might do this if you plan to use the channel to play a single sound **resource** that itself specifies the sound’s data type.

`init`

The desired initialization parameters for the channel. If you cannot determine what types of sounds you will be playing on the channel, pass 0 in this parameter. To specify a sound output device other than the current sound output device, pass the value `kUseOptionalOutputDevice` in the `synth` parameter and the signature of the desired sound output device component in the `init` parameter. Only sounds defined by wave-table data and sampled-sound data currently use the `init` parameter.

`userRoutine`

A pointer to a `SndCallbackProc` (III–2147) callback that the Sound Manager executes whenever it receives a `callbackCmd` command; see “Sound Commands” (IV–2691). If you pass `NIL`, then any `callbackCmd` commands sent to this channel are ignored.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

synth Constants

`squareWaveSynth`

Square wave data.

`waveTableSynth`

Wave table data.

`sampledSynth`

Sampled sound data.

kUseOptionalOutputDevice

Data is going to a sound output device other than the current device. The ability to redirect output away from the current sound output device is intended for use by specialized applications that need to use a specific sound output device. In general, your application should always send sound to the current sound output device selected by the user.

Discussion

This function internally allocates memory to store a queue of sound commands. Regardless of whether or not you allocate your own `SndChannel` structure through the `chan` parameter, the Sound Manager allocates memory for the sound command queue internally.

Special Considerations

Because this function allocates memory, you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier. In Sound Manager versions earlier than version 3.0, only one data type can be produced at any one time. On such platforms, this function may fail if you attempt to open a channel specifying a data type other than the one currently being played.

Programming Info

C interface file: `Sound.h`

Related Java Methods

`quicktime.sound.SndChannel.SndChannel()`

SndPauseFilePlay

Deprecated.

```
OSErr SndPauseFilePlay (
    SndChannelPtr    chan );
```

Version Notes

Introduced in QuickTime 3 or earlier. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: `Sound.h`

SndPlay

SndPlay

Plays a sound **resource** that your application has loaded into memory.

```
OSErr SndPlay (
    SndChannelPtr    chan,
    SndListHandle     sndHandle,
    Boolean           async );
```

chan

A pointer to a valid `SndChannel` (IV–2440) structure. You can pass `NIL` instead of a pointer to a sound channel if you want the Sound Manager to internally allocate a sound channel in your application’s heap zone.

sndHandle

A handle to the sound data to play, which is expected to have the structure of a format 1 or format 2 ‘snd’ **resource**. You can pass a handle to data created by calling `SndRecord` (III–1843) as well as a handle to an actual ‘snd’ resource that you have loaded into memory. To use sampled-sound data with a format 1 ‘snd’ resource, the resource must include a `bufferCmd` command. If a format 1 ‘snd’ resource does not specify which type of sound data is to be played, this function defaults to square-wave data. It also supports format 2 ‘snd’ resources using sampled-sound data and a `bufferCmd` command.

async

A `Boolean` value that indicates whether the sound should be played asynchronously (`TRUE`) or synchronously (`FALSE`). This parameter is ignored (and the sound plays synchronously) if `NIL` is passed in the *chan* parameter.

function result Returns `resProblem` if the sound **resource** has not yet been loaded.
 Returns `noErr` if there is no error.

Discussion

All commands and data referenced by *sndHandle* are sent to the channel. If you supply a sound channel pointer in the *chan* parameter, you can play the sound asynchronously. When a sound is played asynchronously, the callback procedure supplied to `SndNewChannel` (III–1839) can be called when a `callbackCmd` command is processed by the channel. The handle you pass in the *sndHdl* parameter must be locked for as long as the sound is playing asynchronously.

Version Notes

Introduced in QuickTime 3 or earlier. In some versions of the Mac OS prior to version 7.0, the `SndPlay` function will not work properly with sound **resources** that specify the sound data type twice. This might happen if a resource specifies that a

sound consists of sampled-sound data and an application does the same when creating a sound channel.

Programming Info

C interface file: Sound.h

Related Java Methods

quicktime.sound.Sound.play(), quicktime.sound.SndChannel.play()

SndPlayDoubleBuffer

Deprecated.

```
OSErr SndPlayDoubleBuffer (
    SndChannelPtr      chan,
    SndDoubleBufferHeaderPtr theParams );
```

Version Notes

Introduced in QuickTime 3 or earlier. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: Sound.h

SndRecord

Records a sound **resource** into memory.

```
OSErr SndRecord (
    ModalFilterUPP      filterProc,
    Point                corner,
    OSType               quality,
    SndListHandle        *sndHandle );
```

filterProc

A pointer to a ModalFilterProc (III-2098) callback that determines how user actions in the sound recording dialog box are filtered. By specifying your own filter function, you can override or add to the default actions of the items in the dialog box. Otherwise, pass NIL.

corner

The horizontal and vertical coordinates of the upper-left corner of the sound recording dialog box, in global coordinates.

SndRecord

quality

A constant (see below) that specifies the desired quality of the recorded sound. The precise meanings of these constants are defined by the sound input device driver. For Apple-supplied drivers, this parameter determines whether the recorded sound is to be compressed, and if so, whether at a 6:1 or a 3:1 ratio.

sndHandle

On entry, a handle to some storage space or `NIL`. If the handle is `NIL`, the Sound Input Manager allocates a handle of the largest amount of space that it can find in your application's heap and returns the handle in this parameter. It resizes the handle when the user clicks the Save button in the sound recording dialog box. If this parameter is not `NIL`, the Sound Input Manager simply stores the recorded data in the location specified by that handle. On return, this parameter contains a handle to a valid sound **resource** (or it is unchanged, if the call did not execute successfully). The recorded data has the structure of a format 1 'snd ' resource and can be stored as a resource or can be played using `SndPlay` (III-1842).

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

quality Constants

siBestQuality

The best quality available. This specification does not compress the sound and provides the best quality output, but at the expense of increased memory use.

siBetterQuality

Quality better than good. The sound is compressed at 3:1 and is suitable for most nonvoice recording.

siGoodQuality

Good quality. The sound is compressed at 6:1 and is suitable for voice recording.

Discussion

This function displays a sound recording dialog box and is always called synchronously. Controls in the dialog box allow the user to start, stop, pause, and resume sound recording, as well as to play back the recorded sound. The dialog box also lists the remaining recording time and the current microphone sound level.

Special Considerations

Because the `SndRecord` function moves memory, you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SndRecordToFile

Deprecated.

```
OSErr SndRecordToFile (
    ModalFilterUPP    filterProc,
    Point              corner,
    OSType             quality,
    short              fRefNum );
```

Version Notes

Introduced in QuickTime 3 or earlier. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: Sound.h

SndSetInfo

Sets information about an open Sound Manager channel.

```
OSErr SndSetInfo (
    SndChannelPtr    chan,
    OSType           selector,
    const void       *infoPtr );
```

chan

A pointer to a valid SndChannel (IV-2440) structure.

selector

A sound input device information selector that specifies the type of information you need. See “Sound Information Selectors” (IV-2693).

infoPtr

A pointer to a buffer in which information should be returned. This buffer must be large enough to contain the type of information specified in the *selector* parameter.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

SndSetSysBeepState

Discussion

This function should be used instead of attempting to communicate directly with sound components.

Special Considerations

This function uses a selector-based interface similar to `SPBSetDeviceInfo` (III–1881), with the same sound information selectors.

Version Notes

Available only with Sound Manager version 3.1 or later. Check for this by calling `SndSoundManagerVersion` (III–1847) to get the installed version number.

Programming Info

C interface file: `Sound.h`

See Also

See `SPBSetDeviceInfo` (III–1881). For the corresponding get function, see `SndGetInfo` (III–1833).

SndSetSysBeepState

Sets the state of the system alert sound.

```
OSErr SndSetSysBeepState (
    short    sysBeepState );
```

`sysBeepState`

One of two constants (see below) that represents the state of the system alert sound.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

sysBeepState Constants

`sysBeepDisable`

The system alert sound is disabled.

`sysBeepEnable`

The system alert sound is enabled.

Discussion

You can use this function to temporarily disable the system alert sound while you play a sound and then enable the alert sound when you are done. If your application disables the system alert sound, be sure to enable it when your application gets a suspend event.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

See Also

For the corresponding get function, see SndGetSysBeepState (III–1833).

SndSoundManagerVersion

Determines the version of the Sound Manager tools available on the user's computer.

```
NumVersion SndSoundManagerVersion ( void );
```

function result A 4-byte version number that contains the same information as a NumVersion (IV–2324) structure.

Discussion

You can use this function to determine if a computer has the enhanced Sound Manager, which is necessary for multichannel sound and for continuous plays from disk.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SndStartFilePlay

Deprecated.

```
OSErr SndStartFilePlay (
    SndChannelPtr    chan,
    short            fRefNum,
    short            resNum,
    long             bufferSize,
```

SndStopFilePlay

```
void
AudioSelectionPtr
FilePlayCompletionUPP
Boolean

*theBuffer,
theSelection,
theCompletion,
async );
```

Version Notes

Introduced in QuickTime 3 or earlier. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: Sound.h

SndStopFilePlay

Deprecated.

```
OSErr SndStopFilePlay (
    SndChannelPtr    chan,
    Boolean          quietNow );
```

Version Notes

Introduced in QuickTime 3 or earlier. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: Sound.h

SoundComponentAddSource

Adds a new sound source to a sound output device component that can mix multiple channel of audio data.

```
ComponentResult SoundComponentAddSource (
    ComponentInstance    ti,
    SoundSource          *sourceID );
```

ti

A component instance that identifies your sound component.

sourceID

On return, a source ID for the newly created source component chain.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function is called by the Sound Manager to create a new sound source. If your sound output device component can mix multiple channels of sound, it needs to define this function. Your component should call `OpenMixerSoundComponent` (II–1132) to create a new instance of the Apple Mixer component. The Apple Mixer component then creates a sound component chain capable of generating the type of data your sound output device component wants to receive. The Apple Mixer also assigns a unique 4-byte source ID that identifies the new sound source and component chain, which you can retrieve by calling this function. This function should then pass that source ID back to the Sound Manager in the `sourceID` parameter.

Special Considerations

Most sound components do not need to implement this function. Only sound components that can handle more than one source of input need to define it. This function is called at noninterrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SoundComponentGetInfo

Gets information about the capabilities of your sound component.

```
ComponentResult SoundComponentGetInfo (
    ComponentInstance    ti,
    SoundSource          sourceID,
    OSType               selector,
    void                 *infoPtr );
```

`ti`

A component instance that identifies your sound component.

`sourceID`

A source ID for a source component chain.

`selector`

A sound input device information selector that specifies the type of information requested. See “Sound Information Selectors” (IV–2693). If your component cannot provide the information specified by the `selector` parameter, it should pass the selector to its source component.

SoundComponentGetSource

`infoPtr`

On output, a pointer to the information requested by the caller.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function returns information about your sound component. If the information occupies 4 or fewer bytes, it should be returned in the location pointed to by the `infoPtr` parameter. If the information is larger than 4 bytes, the `infoPtr` parameter is a pointer to a `SoundInfoList` (IV–2453) structure. This structure consists of a count and a handle to a variable-sized array. The data type of the array elements depends on the kind of information being returned. For example, the selector `siSampleSizeAvailable` indicates that you should return a list of the sample sizes your component can support. You return the information by passing back, in the `infoPtr` parameter, a pointer to an integer followed by a handle to an array of integers.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding set function, see `SoundComponentSetInfo` (III–1856).

SoundComponentGetSource

Determines your component’s source component.

```
ComponentResult SoundComponentGetSource (  
    ComponentInstance    ti,  
    SoundSource          sourceID,  
    ComponentInstance    *source );
```

`ti`

A component instance that identifies your sound component.

`sourceID`

A source ID for the source component chain created by the Apple Mixer.

`source`

A component instance that identifies your source component. This should be the source component instance your component was passed when the Sound Manager called your `SoundComponentSetSource` (III–1858) function. In the

unlikely event that your component does not have a source, you should return `NIL`.

function result Your `SoundComponentGetSource` function should return `noErr` if successful or an appropriate result code otherwise. See “Error Codes” (IV–2663).

Discussion

This function is called by the Sound Manager to retrieve your component’s source component instance. In general, all sound components have sources, except for the source at the beginning of the source component chain. A sound output device component is always connected directly to an instance of the Apple Mixer. Accordingly, a sound output device component should return a component instance of the Apple Mixer in the source parameter and a source ID in the `sourceID` parameter. A utility component can ignore the `sourceID` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding set function, see `SoundComponentSetSource` (III–1858).

SoundComponentGetSourceData

Lets a sound output device component tell its source component when it needs more data.

```
ComponentResult SoundComponentGetSourceData (
    ComponentInstance    ti,
    SoundComponentDataPtr *sourceData );
```

ti

A component instance that identifies your sound component.

sourceData

On output, a pointer to a `SoundComponentData` (IV–2448) structure that specifies the type and location of the data your component has processed.

function result Your `SoundComponentGetSourceData` function should return `noErr` if successful or an appropriate result code otherwise. See “Error Codes” (IV–2663). If your component cannot generate any more data, it should set the `sampleCount` field of the `SoundComponentData`

SoundComponentInitOutputDevice

(IV–2448) structure to 0 and return noErr.

Discussion

This function is called when the sound component immediately following your sound component in the sound component chain needs more data. Your function should generate a new block of audio data, fill out a `SoundComponentData` (IV–2448) structure describing the format and location of that data, and then return the address of that structure in the `sourceData` parameter. Your function might itself need to get more data from its source component. To do this, it calls its source component’s `SoundComponentGetSourceData` function.

Special Considerations

Sound output device components do not need to implement this function, but all utility components must implement it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SoundComponentInitOutputDevice

Allows a sound output device component to configure associated hardware devices.

```
ComponentResult SoundComponentInitOutputDevice (  
    ComponentInstance    ti,  
    long                 actions );
```

ti

A component instance that identifies your sound component.

actions

A set of flags. This parameter is currently unused.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

Every sound output device component must implement this function. It is called by the Sound Manager at noninterrupt time to allow your sound output device component to perform any hardware-specific initialization. You should perform any necessary initialization that was not already performed in your `OpenComponent` (II–1130) function. Note that your `OpenComponent` function cannot assume that the appropriate hardware is available. As a result, the Sound Manager calls this

function when it is safe to communicate with your audio hardware. You can call `OpenMixerSoundComponent` (II–1132) to create a single sound component chain.

Special Considerations

This function is always called at noninterrupt time. All other component-defined routines might be called at interrupt time. Accordingly, this function should handle any remaining memory allocation needed by your component and it should lock down any relocatable blocks your component will access.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

S

SoundComponentPauseSource

Pauses the playing of sounds in one or more sound channels of your sound output device component.

```
ComponentResult SoundComponentPauseSource (
    ComponentInstance  ti,
    short              count,
    SoundSource        *sources );
```

ti

A component instance that identifies your sound component.

count

The number of source IDs in the array pointed to by the *sources* parameter.

sources

An array of source IDs.

function result Your `SoundComponentPauseSource` function should return `noErr` if successful or an appropriate result code otherwise. See “Error Codes” (IV–2663). You should return `noErr` even if no sounds are playing in the specified channels.

Discussion

This function is called by the Sound Manager to pause the playing of the sounds originating from the sound sources specified by the *sources* parameter. Your function should stop sending data from those sources to the associated sound output device. Because this function might be called to resume playing sounds, you

SoundComponentPlaySourceBuffer

should not flush any data. If your component supports only one sound source, you can ignore the sources parameter.

Special Considerations

Every sound output device component must implement this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SoundComponentPlaySourceBuffer

Starts a new sound playing through your sound output device component.

```
ComponentResult SoundComponentPlaySourceBuffer (
    ComponentInstance    ti,
    SoundSource          sourceID,
    SoundParamBlockPtr  pb,
    long                 actions );
```

ti

A component instance that identifies your sound component.

sourceID

A source ID for a source component chain.

pb

A pointer to a SoundParamBlock (IV–2454) structure.

actions

A set of 32 bit flags (see below) that describe the actions to be taken when preparing to play the source data.

function result Your SoundComponentPlaySourceBuffer function should return noErr if successful or an appropriate result code otherwise. See “Error Codes” (IV–2663).

actions Constants

kSourcePaused

If this bit is 1, the component chain is configured to play the specified sound but the playback is initially paused. In this case, your SoundComponentStartSource function must be called to begin playback. If this bit is 0, the playback begins immediately once the component chain is set up and configured.

kPassThrough

If this bit is 1, the Sound Manager passes all data through to the sound output device component unmodified. A sound output device component that can handle any sample rate and sound format described in a sound parameter block should set this bit to 1.

kNoSoundComponentChain

If this bit is 1, the Sound Manager does not construct a component chain for processing the sound data.

Discussion

This function is called by the Sound Manager to start a new sound playing. The sound parameter block pointed to by the `pb` parameter specifies the sound to be played. That parameter block should be passed successively to all sound components in the chain specified by the `sourceID` parameter. This allows the components to determine their output formats and playback settings and to prepare for a subsequent call to their `SoundComponentGetSourceData (III-1851)` function. It also allows a sound output device component to prepare for starting up its associated hardware.

Special Considerations

Every sound component must implement this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SoundComponentRemoveSource

Removes sound sources from your sound output device component.

```
ComponentResult SoundComponentRemoveSource (
    ComponentInstance  ti,
    SoundSource        sourceID );
```

ti

A component instance that identifies your sound component.

sourceID

A source ID for the source component chain to be removed.

function result Your `SoundComponentRemoveSource` function should return `noErr` if successful or an appropriate result code otherwise See “Error Codes”

SoundComponentSetInfo

(IV–2663).

Discussion

This function is called by the Sound Manager to remove the existing sound source specified by the `sourceID` parameter. Your component should do whatever is necessary to invalidate that source and then call `SoundComponentRemoveSource` (III–1855).

Special Considerations

Every sound output device component that implements `SoundComponentAddSource` (III–1848) must also implement this function. It is always called at noninterrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SoundComponentSetInfo

Modifies the settings of your sound component.

```
ComponentResult SoundComponentSetInfo (
    ComponentInstance  ti,
    SoundSource        sourceID,
    OSType             selector,
    void               *infoPtr );
```

`ti`

A component instance that identifies your sound component.

`sourceID`

A source ID for a source component chain.

`selector`

A sound input device information selector that specifies the type of information to be set. See “Sound Information Selectors” (IV–2693). If your component cannot provide the information specified by the `selector` parameter, it should pass the selector to its source component.

`infoPtr`

A pointer to the information your component is to use to modify its settings. If the information associated with the `selector` parameter occupies 4 or fewer bytes, it is passed on the stack, in the `infoPtr` parameter itself. Otherwise, the `infoPtr` parameter is a pointer to a `SoundInfoList` (IV–2453) structure.

function result Your `SoundComponentSetInfo` function should return `noErr` if successful or an appropriate result code otherwise. See “Error Codes” (IV–2663).

Discussion

This function is called by the Sound Manager to set one of the settings for your component, as specified by the `selector` parameter.

Special Considerations

Every sound component must implement this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding get function, see `SoundComponentGetInfo` (III–1849).

SoundComponentSetOutput

Queries the Apple Mixer component to indicate the type of data a sound output device component expects to receive.

```
ComponentResult SoundComponentSetOutput (
    ComponentInstance      ti,
    SoundComponentDataPtr  requested,
    SoundComponentDataPtr  *actual );
```

ti

A component instance that identifies your sound component.

requested

A pointer to a `SoundComponentData` (IV–2448) structure that specifies the type of the data your component expects to receive.

actual

Currently unused.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function can be called by a sound output device component to specify the kind of audio data the output device component wants to receive. The Apple Mixer uses that information to determine the type of sound component chain it needs to construct in order to deliver that kind of audio data to your sound output device

SoundComponentSetSource

component. For example, if your sound output device is able to accept 16-bit samples, the Sound Manager doesn't need to convert 16-bit audio data into 8-bit data. The following lines of code illustrate how the sound output device component for the Apple Sound Chip (ASC) might call this function in the Apple Mixer:

```
// SoundComponentSetOutput coding example
myDataRec.flags = 0;                      /*ignored here*/
myDataRec.format = kOffsetBinary;         /*ASC needs offset binary*/
myDataRec.sampleRate = rate22khz;        /*ASC needs 22 kHz samples*/
myDataRec.sampleSize = 8;                 /*ASC needs 8-bit data*/
myDataRec.numChannels = 2;                /*ASC can do stereo*/
myDataRec.sampleCount = 1024;             /*ASC uses a 1K FIFO*/
myErr = SoundComponentSetOutput(mySource, &myDataRec, &myActual);
```

Special Considerations

Only the Apple Mixer component needs to implement this function. In general, however, a sound output device component shouldn't need to call this function. Instead, it can indicate the type of data it expects to receive when it calls `OpenMixerSoundComponent` (II-1132). This function is intended for sophisticated sound output device components that might want to reinitialize the Apple Mixer.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SoundComponentSetSource

Identifies your sound component's source component.

```
ComponentResult SoundComponentSetSource (
    ComponentInstance  ti,
    SoundSource        sourceID,
    ComponentInstance  source );
```

`ti`

A component instance that identifies your sound component.

`sourceID`

A source ID for the source component chain created by the Apple Mixer.

`source`

A component instance that identifies your source component.

function result Your SoundComponentSetSource function should return noErr if successful or an appropriate result code otherwise. See “Error Codes” (IV–2663).

Discussion

This function is called by the Sound Manager to identify to your sound component the sound component that is its source. Your component uses that information when it needs to obtain more data from its source; usually, by calling its SoundComponentGetSourceData (III–1851) function. Because a sound output device component is always connected directly to one or more instances of the Apple Mixer, this function needs to be implemented only by utility components (that is, components that perform modifications on sound data). Utility components are linked together into a chain of sound components, each link of which has only one input source. As a result, a utility component can usually ignore the sourceID parameter passed to it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

See Also

For the corresponding get function, see SoundComponentGetSource (III–1850).

SoundComponentStartSource

Starts playing sounds in one or more sound channels.

```
ComponentResult SoundComponentStartSource (
    ComponentInstance  ti,
    short              count,
    SoundSource        *sources );
```

ti

A component instance that identifies your sound component.

count

The number of source IDs in the array pointed to by the source parameter.

sources

An array of source IDs.

function result Your SoundComponentStartSource function should return noErr if successful or an appropriate result code otherwise. See “Error

SoundComponentStopSource

Codes” (IV–2663). You should return `noErr` even if no sounds are playing in the specified channels.

Discussion

This function is called by the Sound Manager to begin playing the sounds originating from the sound sources specified by the `sources` parameter. Your function should start (or resume) sending data from those sources to the associated sound output device. If your component supports only one sound source, you can ignore the `sources` parameter.

Special Considerations

Every sound output device component must implement this function. It can be called at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SoundComponentStopSource

Stops playing sounds in one or more sound channels.

```
ComponentResult SoundComponentStopSource (
    ComponentInstance    ti,
    short                count,
    SoundSource          *sources );
```

ti

A component instance that identifies your sound component.

count

The number of source IDs in the array pointed to by the `source` parameter.

sources

An array of source IDs.

function result Your `SoundComponentStopSource` function should return `noErr` if successful or an appropriate result code otherwise. See “Error Codes” (IV–2663). You should return `noErr` even if no sounds are playing in the specified channels.

Discussion

Your `SoundComponentStopSource` function is called by the Sound Manager to stop the sounds originating from the sound sources specified by the `sources` parameter.

Your function should stop sending data from those sources to the associated sound output device. In addition, your `SoundComponentStopSource` function should flush any data from the specified sound sources that it's caching. If your component supports only one sound source, you can ignore the `sources` parameter.

Special Considerations

Every sound output device component must implement this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

S

SoundConverterBeginConversion

Starts a sound conversion process.

```
OSErr SoundConverterBeginConversion (
    SoundConverter    sc );
```

`sc`

The sound converter identifier returned by `SoundConverterOpen` (III–1867).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function resets all state information to default values in preparation for a new input buffer.

Special Considerations

This routine can be called at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SoundConverterClose

Closes a sound converter.

```
OSErr SoundConverterClose (
    SoundConverter    sc );
```

SoundConverterConvertBuffer

sc

The sound converter identifier returned by SoundConverterOpen (III–1867).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SoundConverterConvertBuffer

Converts a buffer of sound data from an input format to an output format.

```
OSErr SoundConverterConvertBuffer (
    SoundConverter    sc,
    const void        *inputPtr,
    unsigned long      inputFrames,
    void              *outputPtr,
    unsigned long      *outputFrames,
    unsigned long      *outputBytes );
```

sc

The sound converter identifier returned by SoundConverterOpen (III–1867).

inputPtr

A pointer to an input sound data buffer.

inputFrames

The number of frames in the buffer pointed to by inputPtr.

outputPtr

A pointer to a buffer where the output data should be placed.

outputFrames

On return, a pointer to the number of frames written into the buffer pointed to by outputPtr.

outputBytes

On return, a pointer to the number of bytes written into the buffer pointed to by outputPtr.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This routine will consume all the data in the input buffer. Depending on the complexity of the conversion, however, not all the converted data may be put in the output buffer right away. This routine is used to flush out all this remaining data before a conversion session is closed. If you are using this routine in conjunction with `SoundConverterGetBufferSizes` (III–1866), it is very important that you do not pass in a value in `inputFrames` larger than the `frames` parameter value returned by `SoundConverterGetBufferSizes`, or you will overflow your output buffer.

Special Considerations

This function can be called at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SoundConverterEndConversion

Ends a sound conversion process.

```
OSErr SoundConverterEndConversion (
    SoundConverter    sc,
    void              *outputPtr,
    unsigned long      *outputFrames,
    unsigned long      *outputBytes );
```

`sc`

The sound converter identifier returned by `SoundConverterOpen` (III–1867).

`outputPtr`

A pointer to a buffer where the last of the output data should be placed. Any data remaining in the sound converter is flushed out and returned here.

`outputFrames`

On return, a pointer to the number of frames written into the buffer pointed to by `outputPtr`.

`outputBytes`

On return, a pointer to the number of bytes written into the buffer pointed to by `outputPtr`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

SoundConverterFillBuffer

Special Considerations

This routine can be called at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SoundConverterFillBuffer

Provides an alternative method for converting sound data that is more predictable and flexible than using SoundConverterConvertBuffer (III–1862).

```
OSErr SoundConverterFillBuffer (
    SoundConverter          sc,
    SoundConverterFillBufferDataUPP fillBufferDataUPP,
    void                   *fillBufferDataRefCon,
    void                   *outputBuffer,
    unsigned long           outputBufferSize,
    unsigned long           *bytesWritten,
    unsigned long           *framesWritten,
    unsigned long           *outputFlags );
```

sc

The sound converter identifier returned by SoundConverterOpen (III–1867).

fillBufferDataUPP

A **Universal Procedure Pointer** that points at a SoundConverterFillBufferDataProc (III–2148) callback, which can provide data to the Sound Converter. See “Universal Procedure Pointers” (IV–2641).

fillBufferDataRefCon

A pointer that is passed to the SoundConverterFillBufferDataProc (III–2148) callback. You can use it to point to a data structure that the callback accesses.

outputBuffer

A pointer to the buffer to write the data into.

outputBufferSize

The size of outputBuffer in bytes. No more than this amount will be written during a single call to SoundConverterFillBuffer.

bytesWritten

On return, a pointer to the number of bytes that were written into outBuffer.

framesWritten

On return, a pointer to the number of sample frames written into outBuffer.

outputFlags

On return, a pointer to a set of bit flags (see below) that indicate the state of the conversion process. These are advisor flags only; they don't guarantee any internal data. You need to keep track of your data.

function result This function can return any Sound Manager error. See “Error Codes” (IV–2663). Returns noErr if there is no error.

outputFlags Constants

kSoundConverterDidntFillBuffer

Set if the converter didn't completely fill the output buffer.

kSoundConverterHasLeftOverData

Set if the converter still has more data to deliver. This indicates that more calls to this function should be made to complete the conversion. If this flag is not set, it doesn't indicate there is necessarily more data in the pipeline.

Discussion

In general, working with this function is much like working with SoundConverterConvertBuffer (III–1862). The differences begin with how the buffering is done. This function will do as much or as little work as is required to satisfy a given request. This means that you can pass in buffers of any size and expect that the Sound Converter will only give you that amount, at most. Moreover, the SoundConverterFillBufferDataProc (III–2148) callback can be called as many times as necessary to fulfill a request. This means that the callback is free to provide data in whatever chunk size it likes. Of course with both sides, the buffer sizes will control how many times you need to request data and there is a certain amount of overhead for each call. You will want to balance this against the performance you require.

Special Considerations

While a call to SoundConverterGetBufferSizes (III–1866) is not required to use this function, it is useful as a guide for non-VBR formats.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Sound.h

See Also

See SoundConverterConvertBuffer (III–1862).

SoundConverterGetBufferSizes

Determines actual input and output buffer sizes for a given target size in a sound conversion process.

```
OSErr SoundConverterGetBufferSizes (
    SoundConverter    sc,
    unsigned long     inputBytesTarget,
    unsigned long     *inputFrames,
    unsigned long     *inputBytes,
    unsigned long     *outputBytes );
```

sc

The sound converter identifier returned by `SoundConverterOpen` (III–1867).

inputBytesTarget

The approximate number of bytes you would like both your input and output buffers to be.

inputFrames

A pointer to the actual number of frames your input buffer must hold.

inputBytes

A pointer to the actual number of bytes your input buffer must hold.

outputBytes

On return, a pointer to the size in bytes required for your output buffer.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is used to make sure your sound conversion buffers will fit the conversion parameters established with `SoundConverterOpen` (III–1867). The returned input and output buffer sizes are rounded up, depending on the format, but they will be very close to the target settings. The input and output sizes may be very different, depending on the input and output formats given in `SoundConverterOpen`. The sizes are calculated assuming you will convert all data in the input buffer and pass it to the output buffer.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SoundConverterGetInfo

Gets information from a sound converter, using sound information selectors.

```
OSErr SoundConverterGetInfo (
    SoundConverter    sc,
    OSType            selector,
    void              *infoPtr );
```

sc

The sound converter identifier returned by `SoundConverterOpen` (III–1867).

selector

A sound information selector that specifies the type of information to be retrieved. See “Sound Information Selectors” (IV–2693).

infoPtr

On return, a pointer to the information. If the information associated with the *selector* parameter occupies 4 or fewer bytes, it is passed on the stack, in the *infoPtr* parameter itself. Otherwise, the *infoPtr* parameter is a pointer to a `SoundInfoList` (IV–2453) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding set function, see `SoundConverterSetInfo` (III–1868).

SoundConverterOpen

Opens a sound converter component to convert sound data from one format to another.

```
OSErr SoundConverterOpen (
    const SoundComponentData *inputFormat,
    const SoundComponentData *outputFormat,
    SoundConverter            *sc );
```

SoundConverterSetInfo

inputFormat

A pointer to a SoundComponentData (IV–2448) structure that defines the input format.

outputFormat

A pointer to a SoundComponentData (IV–2448) structure that defines the desired output format.

sc

On return, a pointer to a sound converter identifier to be passed to other sound converter functions.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

See Also

During the conversion process, you will call SoundConverterOpen, SoundConverterClose (III–1861), SoundConverterBeginConversion (III–1861) and SoundConverterEndConversion (III–1863). To implement the conversion process, see SoundConverterConvertBuffer (III–1862), SoundConverterFillBuffer (III–1864), and SoundConverterGetBufferSizes (III–1866).

SoundConverterSetInfo

Sets the configuration of a sound converter, using sound information selectors.

```
OSErr SoundConverterSetInfo (
    SoundConverter    sc,
    OSType            selector,
    void              *infoPtr );
```

sc

The sound converter identifier returned by SoundConverterOpen (III–1867).

selector

A sound information selector that specifies the type of information to be set. See “Sound Information Selectors” (IV–2693).

infoPtr

A pointer to the information to be set. If the information associated with the `selector` parameter occupies 4 or fewer bytes, pass it in the `infoPtr` parameter itself. Otherwise, pass a pointer to a `SoundInfoList` (IV–2453) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding get function, see `SoundConverterGetInfo` (III–1867).

SoundManagerGetInfo

Retrieves information about the Sound Manager.

```
OSErr SoundManagerGetInfo (
    OSType    selector,
    void      *infoPtr );
```

selector

A sound information selector that specifies the type of information to be retrieved. See “Sound Information Selectors” (IV–2693).

infoPtr

On return, a pointer to the information. If the information associated with the `selector` parameter occupies 4 or fewer bytes, it is passed on the stack, in the `infoPtr` parameter itself. Otherwise, the `infoPtr` parameter is a pointer to a `SoundInfoList` (IV–2453) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding set function, see `SoundManagerSetInfo` (III–1870).

SoundManagerSetInfo

SoundManagerSetInfo

Sets information about the Sound Manager.

```
OSErr SoundManagerSetInfo (
    OSType      selector,
    const void   *infoPtr );
```

selector

A sound information selector that specifies the type of information to be set. See “Sound Information Selectors” (IV–2693).

infoPtr

A pointer to the information to be set. If the information associated with the *selector* parameter occupies 4 or fewer bytes, pass it in the *infoPtr* parameter itself. Otherwise, pass a pointer to a `SoundInfoList` (IV–2453) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding get function, see `SoundManagerGetInfo` (III–1869).

SPBBytesToMilliseconds

Determines the maximum duration of a recording that can fit in a buffer of a certain size.

```
OSErr SPBBytesToMilliseconds (
    long    inRefNum,
    long    *byteCount );
```

inRefNum

The device reference number of a sound input device, returned by `SPBOpenDevice` (III–1876).

byteCount

On entry, a buffer size in bytes. On return, the number of milliseconds of recording on the device specified by the *inRefNum* parameter that would be

necessary to fill a buffer of such a size, given the input device's current sample rate, sample size, number of channels, and compression factor.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

Related Java Methods

quicktime.sound.SPBCloseDevice.bytesToMilliseconds()

SPBCloseDevice

Closes a sound input device.

```
OSErr SPBCloseDevice (
    long    inRefNum );
```

inRefNum

The device reference number of a sound input device, returned by SPBOpenDevice (III–1876).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Special Considerations

Because this function moves or purges memory, you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SPBGetDeviceInfo

Gets information about the settings of a sound input device.

```
OSErr SPBGetDeviceInfo (
    long    inRefNum,
```

SPBGetDeviceInfo

```
OSType      infoType,  
void        *infoData );
```

inRefNum

The device reference number of the sound input device, as obtained from `SPBOpenDevice` (III–1876).

infoType

A sound input device information selector that specifies the type of information you need. See “Sound Information Selectors” (IV–2693).

infoData

A pointer to a buffer in which information should be returned. This buffer must be large enough to contain the type of information specified in the `infoType` parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function uses a selector-based interface similar to `SndGetInfo` (III–1833), with the same sound information selectors.

Special Considerations

Because this function might move memory, you should not call it at interrupt time. Check the selector description of the selector you want to use, to see if it moves memory, before calling this function. Most of the selectors do not move memory and are therefore safe to use at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

Related Java Methods

```
quicktime.sound.SPBDevice.setLevelMeterOnOff(),  
quicktime.sound.SPBDevice.getLevelMeterOnOff(),  
quicktime.sound.SPBDevice.getLevelMeterLevel(),  
quicktime.sound.SPBDevice.setAutomaticGainControl(),  
quicktime.sound.SPBDevice.getAutomaticGainControl(),  
quicktime.sound.SPBDevice.setInputGain(),  
quicktime.sound.SPBDevice.getInputGain(),  
quicktime.sound.SPBDevice.setInputSource(),  
quicktime.sound.SPBDevice.setInputSource(),  
quicktime.sound.SPBDevice.getInputSourceNames(),  
quicktime.sound.SPBDevice.getChannelAvailable(),
```

```

quicktime.sound.SPBDevice.getNumberChannels(),
quicktime.sound.SPBDevice.setNumberChannels(),
quicktime.sound.SPBDevice.getPlayThruOnOff(),
quicktime.sound.SPBDevice.setPlayThruOnOff(),
quicktime.sound.SPBDevice.setSampleRate(),
quicktime.sound.SPBDevice.getSampleRate(),
quicktime.sound.SPBDevice.getSampleRateAvailable(),
quicktime.sound.SPBDevice.setSampleSize(),
quicktime.sound.SPBDevice.getSampleSize(),
quicktime.sound.SPBDevice.getSampleSizeAvailable(),
quicktime.sound.SPBDevice.setStereoInputGain(),
quicktime.sound.SPBDevice.getStereoInputGainLeft(),
quicktime.sound.SPBDevice.getStereoInputGainRight(),
quicktime.sound.SPBDevice.setCompressionType(),
quicktime.sound.SPBDevice.getCompressionType(),
quicktime.sound.SPBDevice.getCompressionAvailable(),
quicktime.sound.SPBDevice.hasOptionsDialog(),
quicktime.sound.SPBDevice.showOptionsDialog()

```

See Also

See `SndGetInfo` (III–1833). For the corresponding set function, see `SPBSetDeviceInfo` (III–1881).

SPBGetIndexedDevice

Helps you generate a list of sound input devices.

```

OSErr SPBGetIndexedDevice (
    short      count,
    Str255     deviceName,
    Handle     *deviceIconHandle );

```

`count`

The index number of the sound input device you wish to obtain information about.

`deviceName`

On return, the name of the sound input device specified by the `count` parameter.

SPBGetRecordingStatus

`deviceIconHandle`

On return, a handle to the icon of the sound input device specified by the `count` parameter. The memory for this icon is allocated automatically, but your application must dispose of it.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error. Returns `siBadSoundInDevice` if the value of the `count` parameter is greater than the number of sound input devices.

Discussion

Your application can create a list of sound input devices by calling this function with the `count` parameter set to 1 and incrementing the `count` parameter by 1 until the function returns `siBadSoundInDevice`.

Special Considerations

Because the Sound In control panel allows the user to select a sound input device, most applications should not use this function. Your application might need to use this function if it allows the user to record from more than one sound input device at once. This function allocates memory, so you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SPBGetRecordingStatus

Obtains recording status information about a sound input device.

```
OSErr SPBGetRecordingStatus (
    long          inRefNum,
    short         *recordingStatus,
    short         *meterLevel,
    unsigned long *totalSamplesToRecord,
    unsigned long *numberOfSamplesRecorded,
    unsigned long *totalMsecsToRecord,
    unsigned long *numberOfMsecsRecorded );
```

`inRefNum`

On return, a pointer to the device reference number of the sound input device, returned by `SPBOpenDevice` (III–1876).

`recordingStatus`

On return, a pointer to the status of the recording. While the input device is recording, this parameter is set to a number greater than 0. When a recording terminates without an error, this parameter is set to 0. When an error occurs during recording or the recording has been terminated by a call to `SPBStopRecording` (III–1883), this parameter is less than 0 and contains an error code.

`meterLevel`

On return, a pointer to the current input signal level, ranging from 0 to 255.

`totalSamplesToRecord`

On return, a pointer to the total number of samples to record, including those samples already recorded.

`numberOfSamplesRecorded`

On return, a pointer to the number of samples already recorded.

`totalMsecsToRecord`

On return, a pointer to the total duration of recording time, including recording time already elapsed, in milliseconds.

`numberOfMsecsRecorded`

On return, a pointer to the amount of recording time that has elapsed, in milliseconds.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

Related Java Methods

`quicktime.sound.SPB.isRecording()`, `quicktime.sound.SPB.meterLevel()`

SPBMillisecondsToBytes

Determines how many bytes a recording of a certain duration will occupy.

```
OSErr SPBMillisecondsToBytes (
    long    inRefNum,
    long    *milliseconds );
```

SPBOpenDevice

`inRefNum`

The device reference number of the sound input device, returned by `SPBOpenDevice` (III–1876).

`milliseconds`

On entry, the duration of the recording in milliseconds. On return, the number of bytes that sampled-sound data would occupy for a recording of the specified duration on the device specified by the `inRefNum` parameter, given the input device’s current sample rate, sample size, number of channels, and compression factor.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

Related Java Methods

`quicktime.sound.SPBDDevice.millisecondsToBytes()`

SPBOpenDevice

Opens a sound input device.

```
OSErr SPBOpenDevice (
    ConstStr255Param  deviceName,
    short              permission,
    long               *inRefNum );
```

`deviceName`

The name of the sound input device to open. You can request that the current default sound input device be opened by passing either a 0-length string or a NIL string.

`permission`

A flag (see below) that indicates whether subsequent operations with that device are to be read / write or read-only. If the device is not already in use, read / write permission is granted; otherwise, only read-only operations are allowed.

`inRefNum`

On return, if the function is successful, a device reference number for the open sound input device.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

permission Constants

`siReadPermission`

Open device for reading only.

`siWritePermission`

Open device for reading and writing. To make any recording requests or to call `SPBSetDeviceInfo` (III–1881), read / write permission must be available.

Discussion

Generally, you should open the default device unless you specifically want to use some other device. You can get a list of the available devices by calling `SPBGetIndexedDevice` (III–1873). If only one sound input device is installed, that device is opened.

Special Considerations

Because this function allocates memory, you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

Related Java Methods

`quicktime.sound.SPBDevice.SPBDevice()`

SPBPauseRecording

Pauses recording from a sound input device.

```
OSErr SPBPauseRecording (
    long    inRefNum );
```

`inRefNum`

The device reference number of the sound input device, returned by `SPBOpenDevice` (III–1876).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

SPBRecord

Discussion

This function pauses recording for the device specified by the `inRefNum` parameter. The recording must be asynchronous for this call to have any effect.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

Related Java Methods

`quicktime.sound.SPB.pauseRecording()`

SPBRecord

Records sound data into memory, either synchronously or asynchronously.

```
OSErr SPBRecord (
    SPBPtr      inParamPtr,
    Boolean     asynchFlag );
```

inParamPtr

A pointer to an SPB (IV–2456) structure. The fields of this structure pass data in and out of the function as described below.

asynchFlag

A Boolean value that specifies whether the recording occurs asynchronously (TRUE) or synchronously (FALSE).

function result This function returns the value that the error field of the SPB (IV–2456) structure contains when recording finishes. See “Error Codes” (IV–2663). It returns `noErr` if there is no error.

SPB structure fields

inRefNum

The device reference number of the sound input device, as obtained from `SPBOpenDevice` (III–1876).

count

On input, the number of bytes to record. If this field indicates a longer recording time than the milliseconds field, then the milliseconds field is ignored. On output, this field indicates the number of bytes actually recorded.

milliseconds

On input, the number of milliseconds to record. If this field indicates a longer recording time than the `count` field, then the `count` field is ignored. On output, this field indicates the number of milliseconds actually recorded.

bufferLength

The number of bytes in the buffer specified by the `bufferPtr` parameter. If this buffer length is too small to contain the amount of sampled-sound data specified in the `count` field and the `milliseconds` field, then recording time is truncated so that the sampled-sound data fits in the buffer.

bufferPtr

A pointer to the buffer for the sampled-sound data, or `NIL` if you wish to record sampled-sound data without saving it. On return, this buffer contains the sampled-sound data, which is interleaved for stereo sound on a sample basis (or on a packet basis if the data is compressed). This buffer contains only sampled-sound data, so if you need a sampled sound header, you should set that up in a buffer before calling this function and then record into the buffer following the sound header.

completionRoutine

A pointer to an `SICompletionProc` (III–2146) callback. This routine is called when the recording terminates, either after you call `SPBStopRecording` (III–1883) or when the prescribed limit is reached. The completion routine is called only for asynchronous recording.

interruptRoutine

A pointer to an `SIInterruptProc` (III–2147) callback. This interrupt routine is called by asynchronous recording devices when their internal buffers are full.

userLong

A long integer; it can be a pointer to a data structure that passes data to your application's `SICompletionProc` or `SIInterruptProc` callbacks.

error

On return, a value greater than 0 while recording unless an error occurs, in which case it contains a value less than 0 that indicates an operating system error. Your application can poll this field to check on the status of an asynchronous recording. If recording terminates without an error, this field contains 0.

unused1

Reserved. You should set this field to 0 before calling this function.

SPBRecordToFile

Discussion

This function starts recording into memory from a device specified in an SPB (IV–2456) structure. The sound data recorded is stored in the buffer specified by the `bufferPtr` and `bufferLength` fields of the structure. Recording lasts the longer of the times specified by the `count` and `milliseconds` fields of the structure, or until the buffer is filled. Recording is asynchronous if the `asynchFlag` parameter is `TRUE` and the specified sound input device supports asynchronous recording.

If the `bufferPtr` field of the SPB structure contains `NIL`, then the `count`, `milliseconds`, and `bufferLength` fields are ignored, and the recording continues indefinitely until you call `SPBStopRecording` (III–1883). In this case, the audio data is not saved anywhere; this feature is useful only if you want to do something in your interrupt routine and do not want to save the audio data. However, if the recording is synchronous and `bufferPtr` is `NIL`, this function returns the result code `siNoBufferSpecified`.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

Related Java Methods

`quicktime.sound.SPB.record()`

SPBRecordToFile

Obsolete.

```
OSErr SPBRecordToFile (
    short      fRefNum,
    SPBPtr     inParamPtr,
    Boolean     asynchFlag );
```

Version Notes

Introduced in QuickTime 3 or earlier. Macintosh Note: Not supported by **CarbonLib**.

Programming Info

C interface file: `Sound.h`

SPBResumeRecording

Resumes recording from a sound input device.

```
OSErr SPBResumeRecording (
    long    inRefNum );
```

inRefNum

The device reference number of the sound input device, returned by `SPBOpenDevice` (III–1876).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Recording on the device specified by the `inRefNum` parameter must previously have been paused by a call to `SPBPauseRecording` (III–1877) for this function to have any effect.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

Related Java Methods

`quicktime.sound.SPB.resumeRecording()`

SPBSetDeviceInfo

Sets information in a sound input device.

```
OSErr SPBSetDeviceInfo (
    long    inRefNum,
    OSType  infoType,
    void    *infoData );
```

inRefNum

The device reference number of the sound input device, returned by `SPBOpenDevice` (III–1876).

infoType

A sound input device information selector that specifies the type of information to be set. See “Sound Information Selectors” (IV–2693).

SPBSignInDevice

infoData

A pointer to a buffer. This buffer can contain information on entry, and information might be returned on return. This buffer must be large enough for the type of information specified in the *infoType* parameter.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Special Considerations

Because this function might move memory, you should be careful about calling it at interrupt time. Check the selector description of the selector you want to use to see if it moves memory. Most of the selectors do not move memory and are therefore safe to use at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

See Also

For the corresponding get function, see `SPBGetDeviceInfo` (III–1871).

SPBSignInDevice

Registers a sound input device with the Sound Input Manager.

```
OSErr SPBSignInDevice (
    short          deviceRefNum,
    ConstStr255Param deviceName );
```

deviceRefNum

The device driver reference number of the sound input device to register.

deviceName

The device’s name as it is to appear to the user in the Sound In control panel. This is usually not the name of the driver used by the Device Manager. Accordingly, the name should be as descriptive as possible.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Discussion

You should call this function after you have already opened your driver by calling normal Device Manager routines.

Special Considerations

Because this function moves or purges memory, you should not call it at interrupt time. You can, however, call it at system startup time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SPBSignOutDevice

Cancels the registration of a device you have previously registered with SPBSignInDevice (III–1882).

```
OSErr SPBSignOutDevice (
    short    deviceRefNum );
```

deviceRefNum

The driver reference number of the device you wish to sign out.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

When this function cancels the registration of a device, the device is unregistered from the Sound Input Manager’s list of available sound input devices and no longer appears in the Sound In control panel.

Special Considerations

Ordinarily, you should not need to use this function. You might use it if your device driver detects that a sound input device is not functioning correctly or has been disconnected. Because this function moves or purges memory, you should not call it at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

SPBStopRecording

Ends a recording session from a sound input device.

SPBVersion

```
OSErr SPBStopRecording (
    long    inRefNum );
```

inRefNum

The device reference number of the sound input device, returned by SPBOpenDevice (III–1876).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The recording process must be asynchronous for this function to have any effect. When you call this function, the `SICompletionProc` (III–2146) callback specified in the `completionRoutine` field of the `SPB` (IV–2456) structure is called and the `error` field of that structure is set to `abortErr`. This structure was set up by the previous call to `SPBRecord` (III–1878). If you are writing a device driver, you will receive a `KillIO` status call.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

Related Java Methods

`quicktime.sound.SPB.stopRecording()`

SPBVersion

Determines the version of the sound input tools available on the user’s computer.

```
NumVersion SPBVersion ( void );
```

function result A `NumVersion` (IV–2324) structure that contains Sound Input Manager version information.

Special Considerations

You can call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

SpriteHitTest

Determines whether a location in a **sprite**'s display coordinate system intersects the sprite.

```
OSErr SpriteHitTest (
    Sprite    theSprite,
    long      flags,
    Point     loc,
    Boolean   *wasHit );
```

theSprite

The **sprite** for this operation.

flags

Specifies flags (see below) that control the hit testing operation.

loc

A point in the **sprite** world's display space to test for the existence of a sprite. You should apply the sprite world's matrix to the point before passing it to this function.

wasHit

A pointer to a Boolean. On return, the value of the Boolean is TRUE if the **sprite** is at the specified location.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See "Error Codes" (IV-2663).

flags Constants

`spriteHitTestBounds`

The specified location must be within the **sprite**'s bounding box.

`spriteHitTestImage`

If both this flag and `spriteHitTestBounds` are set, the specified location must be within the shape of the sprite's image.

`spriteHitTestInvisibleSprites`

This flag enables invisible **sprites** to be hit tested.

`spriteHitTestIsClick`

This flag is for codecs that want mouse events, such as the Ripple codec.

`spriteHitTestLocInDisplayCoordinates`

Set this flag if you want to pass a display coordinate point in the `loc` parameter.

SpriteMediaCountImages

Discussion

This function is useful for hit testing a subset of the **sprites** in a sprite world and for detecting multiple hits for a single location.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Creating and Manipulating Sprites” (V–2901)

Related Java Methods

`quicktime.std.anim.Sprite.hitTest()`

SpriteMediaCountImages

Retrieves the number of images that currently exist in a **sprite** track.

```
ComponentResult SpriteMediaCountImages (  
    MediaHandler    mh,  
    short           *numImages );
```

mh

The **sprite** media handler for this operation.

numImages

A pointer to a short integer. On return, this integer contains the number of images for the **sprite** media’s current time.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

This function determines the number of images that currently exist based on the **key frame** that is in effect.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Movie Sprites” (V–2902)

Related Java Methods

`quicktime.std.movies.media.SpriteMediaHandler.countImages()`

SpriteMediaCountSprites

Retrieves the number of **sprites** that currently exist in a sprite track.

```
ComponentResult SpriteMediaCountSprites (
    MediaHandler    mh,
    short           *numSprites );
```

mh

The **sprite** media handler for this operation.

numSprites

A pointer to a short integer. On return, this integer contains the number of **sprites** for the sprite media's current time.

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Discussion

This function determines the number of **sprites** that currently exist based on the **key frame** that is in effect.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Managing Movie Sprites” (V-2902)

Related Java Methods

quicktime.std.movies.media.SpriteMediaHandler.countSprites()

SpriteMediaDisposeSprite

Disposes of memory allocated for a **sprite**.

```
ComponentResult SpriteMediaDisposeSprite (
    MediaHandler    mh,
    QTAtomID        spriteID );
```

mh

The **sprite** media handler for this operation.

spriteID

The ID of the **sprite** for this operation.

SpriteMediaGetActionVariable

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

SpriteMediaGetActionVariable

Returns the value of the **sprite** track variable with the specified ID.

```
ComponentResult SpriteMediaGetActionVariable (
    MediaHandler    mh,
    QTAtomID        variableID,
    float            *value );
```

mh

The **sprite** media handler for this operation.

variableID

A variable ID of the **sprite** variable.

value

A pointer to a floating-point value. If the specified variable has never been set, the value is set to 0 and the error `cannotFindAtomErr` is returned.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Wired Sprites” (V–2903)

Related Java Methods

`quicktime.std.movies.media.SpriteMediaHandler.getActionVariable()`

See Also

For the corresponding set function, see `SpriteMediaSetActionVariable` (III–1901).

SpriteMediaGetActionVariableAsString

Undocumented

```
ComponentResult SpriteMediaGetActionVariableAsString (
    MediaHandler    mh,
    QTAtomID        variableID,
    Handle          *theCString );
```

mh

The **sprite** media handler for this operation.

variableID

A variable ID of the **sprite** variable.

theCString

A pointer to a handle to a C string.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

SpriteMediaGetDisplayedSampleNumber

Retrieves the number of the **sprite** media sample that is currently being displayed.

```
ComponentResult SpriteMediaGetDisplayedSampleNumber (
    MediaHandler    mh,
    long            *sampleNum );
```

mh

The **sprite** media handler for this operation.

sampleNum

A pointer to a long integer. On return, this integer contains the number of the sample that is currently being displayed.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

SpriteMediaGetImageName

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Movie Sprites” (V–2902)

Related Java Methods

`quicktime.std.movies.media.SpriteMediaHandler.getDisplayedSampleNumber()`

SpriteMediaGetImageName

Returns the name of the image with the specified index from the current **key frame** sample.

```
ComponentResult SpriteMediaGetImageName (
    MediaHandler    mh,
    short           imageIndex,
    Str255          imageName );
```

mh

The **sprite** media handler for this operation.

imageIndex

The index of the image whose image name is to be retrieved. This value must be between 1 and the number of available images. You can determine how many images are available by calling `SpriteMediaCountImages` (III–1886).

imageName

Returns a Pascal string with the image name of the image, or an empty string if the image is unnamed.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Movie Sprites” (V–2902)

Related Java Methods

`quicktime.std.movies.media.SpriteMediaHandler.getImageName()`

SpriteMediaGetIndImageDescription

Retrieves an image description for a specified image in a **sprite** track.

```
ComponentResult SpriteMediaGetIndImageDescription (
    MediaHandler      mh,
    short             imageIndex,
    ImageDescriptionHandle  imageDescription );
```

mh

The **sprite** media handler for this operation.

imageIndex

The index of the image whose image description is to be retrieved. This value must be between 1 and the number of available images. You can determine how many images are available by calling `SpriteMediaCountImages` (III–1886).

imageDescription

Specifies an image description handle. On return, this handle contains the `ImageDescription` (IV–2285) structure that describes the specified image. This handle must be unlocked; the function resizes the handle if necessary.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Movie Sprites” (V–2902)

Related Java Methods

`quicktime.std.movies.media.SpriteMediaHandler.getIndImageDescription()`

SpriteMediaGetIndImageProperty

Returns a property value for a **sprite** image specified by an index.

```
ComponentResult SpriteMediaGetIndImageProperty (
    MediaHandler      mh,
    short             imageIndex,
    long              imagePropertyType,
    void              *imagePropertyValue );
```

SpriteMediaGetIndImageProperty

mh

The **sprite** media handler for this operation.

imageIndex

The index of the image whose property value is to be retrieved. This value must be between 1 and the number of available images. You can determine how many images are available by calling `SpriteMediaCountImages` (III–1886).

imagePropertyType

The property whose value should be retrieved (see below).

imagePropertyValue

A pointer to a variable that will hold the selected property value on return.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

imagePropertyType Constants

kSpritePropertyMatrix

The *propertyValue* parameter returns the **sprite**’s `MatrixRecord` (IV–2304) structure.

kSpritePropertyImageDescription

The *propertyValue* parameter returns an `ImageDescriptionHandle`, which is a handle to the **sprite**’s `ImageDescription` (IV–2285) structure.

kSpritePropertyImageDataPtr

The *propertyValue* parameter returns a `Ptr`, which is a pointer to the **sprite**’s image data.

kSpritePropertyVisible

The *propertyValue* parameter returns a `Boolean` that is `TRUE` if the **sprite** is visible, `FALSE` otherwise.

kSpritePropertyLayer

The *propertyValue* parameter returns the **sprite**’s layer number.

kSpritePropertyGraphicsMode

The *propertyValue* parameter returns the **sprite**’s `ModifierTrackGraphicsModeRecord` (IV–2311).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Movie Sprites” (V–2902)

Related Java Methods

```
quicktime.std.movies.media.SpriteMediaHandler.getSpriteImageGroupID(),
quicktime.std.movies.media.SpriteMediaHandler.getSpriteImageRegistrationPoint()
```

SpriteMediaGetProperty

Gets a **sprite** property; superseded by SpriteMediaGetSpriteProperty (III–1896).

```
ComponentResult SpriteMediaGetProperty (
    MediaHandler    mh,
    short           spriteIndex,
    long            propertyType,
    void            *propertyValue );
```

mh

The **sprite** media handler for this operation.

spriteIndex

The index of the **sprite** for this operation.

propertyType

The property whose value should be retrieved (see below).

propertyValue

A pointer to a variable that will hold the selected property value on return.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

propertyType Constants

kSpritePropertyMatrix

The *propertyValue* parameter returns the **sprite**’s MatrixRecord (IV–2304) structure.

kSpritePropertyImageDescription

The *propertyValue* parameter returns an ImageDescriptionHandle, which is a handle to the **sprite**’s ImageDescription (IV–2285) structure.

kSpritePropertyImageDataPtr

The *propertyValue* parameter returns a Ptr, which is a pointer to the **sprite**’s image data.

SpriteMediaGetSpriteActionsForQTEvent

kSpritePropertyVisible

The `propertyValue` parameter returns a Boolean that is TRUE if the **sprite** is visible, FALSE otherwise.

kSpritePropertyLayer

The `propertyValue` parameter returns the **sprite**'s layer number.

kSpritePropertyGraphicsMode

The `propertyValue` parameter returns the **sprite**'s `ModifierTrackGraphicsModeRecord` (IV–2311).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

See Also

For the corresponding set function, see `SpriteMediaSetProperty` (III–1903).

SpriteMediaGetSpriteActionsForQTEvent

Gets the **sprite** action atom for an event.

```
ComponentResult SpriteMediaGetSpriteActionsForQTEvent (
    MediaHandler      mh,
    QTEventRecordPtr  event,
    QTAtomID          spriteID,
    QTAtomContainer   *container,
    QTAtom            *atom );
```

`mh`

The **sprite** media handler for this operation.

`event`

A pointer to a `QTEventRecord` (IV–2353) structure.

`spriteID`

The ID of the **sprite** for this operation.

`container`

A pointer to a QT atom container.

`atom`

A pointer to a QT atom in the container.

function result You can access Movie Toolbox error returns through `GetMoviesError`

(I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

SpriteMediaGetSpriteName

Returns the name of the **sprite** with the specified ID from the currently displayed sample.

```
ComponentResult SpriteMediaGetSpriteName (
    MediaHandler    mh,
    QTAtomID        spriteID,
    Str255          spriteName );
```

mh

The **sprite** media handler for this operation.

spriteID

The **sprite** ID of the sprite name.

spriteName

Returns a Pascal string with the name of the **sprite** or an empty string if the sprite is unnamed.

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Movie Sprites” (V-2902)

Related Java Methods

`quicktime.std.movies.media.SpriteMediaHandler.getSpriteName()`

SpriteMediaGetSpriteProperty

SpriteMediaGetSpriteProperty

Retrieves the value of the specified **sprite** property.

```
ComponentResult SpriteMediaGetSpriteProperty (  
    MediaHandler      mh,  
    QTAtomID          spriteID,  
    long              propertyType,  
    void              *propertyValue );
```

mh

The **sprite** media handler for this operation.

spriteID

The ID of the **sprite** for this operation.

propertyType

The property whose value should be retrieved (see below).

propertyValue

On return, a pointer to the value of the property; the data type of that value depends on the property.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

propertyType Constants

kSpritePropertyMatrix

The *propertyValue* parameter returns the **sprite**’s `MatrixRecord` (IV–2304) structure.

kSpritePropertyImageDescription

The *propertyValue* parameter returns an `ImageDescriptionHandle`, which is a handle to the **sprite**’s `ImageDescription` (IV–2285) structure.

kSpritePropertyImageDataPtr

The *propertyValue* parameter returns a `Ptr`, which is a pointer to the **sprite**’s image data.

kSpritePropertyVisible

The *propertyValue* parameter returns a `Boolean` that is `TRUE` if the **sprite** is visible, `FALSE` otherwise.

kSpritePropertyLayer

The *propertyValue* parameter returns the **sprite**’s layer number.

kSpritePropertyGraphicsMode

The `propertyValue` parameter returns the **sprite's** `ModifierTrackGraphicsModeRecord` (IV–2311).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Movie Sprites” (V–2902)

Related Java Methods

```
quicktime.std.movies.media.SpriteMediaHandler.getMatrix(),
quicktime.std.movies.media.SpriteMediaHandler.getVisible(),
quicktime.std.movies.media.SpriteMediaHandler.getLayer(),
quicktime.std.movies.media.SpriteMediaHandler.getGraphicsMode(),
quicktime.std.movies.media.SpriteMediaHandler.getImageIndex()
```

See Also

For the corresponding set function, see `SpriteMediaSetSpriteProperty` (III–1904).

SpriteMediaHitTestAllSprites

Determines whether any **sprites** are at a specified location.

```
ComponentResult SpriteMediaHitTestAllSprites (
    MediaHandler    mh,
    long            flags,
    Point           loc,
    QTAtomID        *spriteHitID );
```

`mh`

The **sprite** media handler for this operation.

`flags`

Specifies flags (see below) that control the hit testing operation.

`loc`

A point in the coordinate system of the **sprite** track's movie to test for the existence of a sprite.

`spriteHitID`

A pointer to a short integer. On return, this integer contains the ID of the frontmost **sprite** at the location specified by `loc`. If no sprite exists at the location, the function sets the value of this parameter to 0.

SpriteMediaHitTestOneSprite

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

flags Constants

`spriteHitTestBounds`

The specified location must be within the **sprite**’s bounding box.

`spriteHitTestImage`

If both this flag and **spriteHitTestBounds** are set, the specified location must be within the shape of the sprite’s image.

`spriteHitTestInvisibleSprites`

This flag enables invisible **sprites** to be hit tested.

`spriteHitTestIsClick`

This flag is for codecs that want mouse events, such as the Ripple codec.

`spriteHitTestLocInDisplayCoordinates`

Set this flag if you want to pass a display coordinate point in the `loc` parameter.

Discussion

You call this function to determine whether any **sprites** exist at a specified location in the coordinate system of a sprite track’s movie. You can pass flags to this function to control the hit testing operation more precisely. For example, you may want the hit test operation to detect a sprite whose bounding box contains the specified location.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Movie Sprites” (V–2902)

Related Java Methods

`quicktime.std.movies.media.SpriteMediaHandler.hitTestAllSprites()`

SpriteMediaHitTestOneSprite

Performs a hit testing operation on the **sprite** specified by a `spriteID`.

```
ComponentResult SpriteMediaHitTestOneSprite (
    MediaHandler    mh,
    QTAtomID        spriteID,
    long            flags,
    Point           loc,
```



```
Boolean *wasHit );
```

mh

The **sprite** media handler for this operation.

spriteID

The **sprite** ID of the sprite.

flags

Flags (see below) that control the hit testing operation.

loc

A point to test for the existence of a **sprite**. The point should be defined in the local coordinates of the sprite track, unless the **spriteHitTestLocInDisplayCoordinates** flag is set.

wasHit

A pointer to a Boolean. If the **sprite** is hit, **wasHit** is set to TRUE; otherwise, it is set to FALSE.

function result You can access Movie Toolbox error returns through **GetMoviesError** (I-492) and **GetMoviesStickyError** (I-494), as well as in the function result. See “Error Codes” (IV-2663).

flags Constants

spriteHitTestBounds

The specified location must be within the **sprite**’s bounding box.

spriteHitTestImage

If both this flag and **spriteHitTestBounds** are set, the specified location must be within the shape of the sprite’s image.

spriteHitTestInvisibleSprites

This flag enables invisible **sprites** to be hit tested.

spriteHitTestIsClick

This flag is for codecs that want mouse events, such as the Ripple codec.

spriteHitTestLocInDisplayCoordinates

Set this flag if you want to pass a display coordinate point in the **loc** parameter.

Discussion

This routine allows you to hit test a **sprite** which is fully or partially covered by other sprites.

Version Notes

Introduced in QuickTime 3 or earlier.

SpriteMediaHitTestSprites

Programming Info

C interface file: Movies.h

Programming summary: “Managing Movie Sprites” (V–2902)

Related Java Methods

quicktime.std.movies.media.SpriteMediaHandler.hitTestOneSprite()

SpriteMediaHitTestSprites

Undocumented

```
ComponentResult SpriteMediaHitTestSprites (  
    MediaHandler    mh,  
    long            flags,  
    Point           loc,  
    short           *spriteHitIndex );
```

mh

The **sprite** media handler for this operation.

flags

Flags (see below) that control the hit testing operation.

loc

A point to test for the existence of a **sprite**.

spriteHitIndex

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

flags Constants

spriteHitTestBounds

The specified location must be within the **sprite**’s bounding box.

spriteHitTestImage

If both this flag and **spriteHitTestBounds** are set, the specified location must be within the shape of the sprite’s image.

spriteHitTestInvisibleSprites

This flag enables invisible **sprites** to be hit tested.

spriteHitTestIsClick

This flag is for codecs that want mouse events, such as the Ripple codec.

`spriteHitTestLocInDisplayCoordinates`

Set this flag if you want to pass a display coordinate point in the `loc` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

SpriteMediaNewSprite

Creates a new **sprite**.

```
ComponentResult SpriteMediaNewSprite (
    MediaHandler          mh,
    QTRuntimeSpriteDescPtr newSpriteDesc );
```

mh

The **sprite** media handler for this operation.

newSpriteDesc

A pointer to a `QTRuntimeSpriteDescStruct` (IV–2363) structure.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

SpriteMediaSetActionVariable

Sets the value of a **sprite** track variable to a specified value.

```
ComponentResult SpriteMediaSetActionVariable (
    MediaHandler    mh,
    QTAtomID        variableID,
    const float      *value );
```

mh

The **sprite** media handler for this operation.

SpriteMediaSetActionVariableToString

variableID

A variable ID of the **sprite** name.

value

A pointer to a floating-point number. The value is passed by reference.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

This function is specific to **sprite** tracks using wired sprites.

Special Considerations

This function is specific to **sprite** tracks using wired sprites.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Wired Sprites” (V–2903)

Related Java Methods

`quicktime.std.movies.media.SpriteMediaHandler.setActionVariable()`

See Also

For the corresponding get function, see `SpriteMediaGetActionVariable` (III–1888).

SpriteMediaSetActionVariableToString

Undocumented

```
ComponentResult SpriteMediaSetActionVariableToString (
    MediaHandler    mh,
    QTAtomID        variableID,
    Ptr              theCString );
```

mh

The **sprite** media handler for this operation.

variableID

A variable ID of the **sprite** variable.

theCString

A pointer to a C string.

function result You can access Movie Toolbox error returns through `GetMoviesError`

(I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

SpriteMediaSetProperty

Sets a **sprite** property; superseded by `SpriteMediaSetSpriteProperty` (III–1904).

```
ComponentResult SpriteMediaSetProperty (
    MediaHandler      mh,
    short             spriteIndex,
    long              propertyType,
    void              *propertyValue );
```

mh

The **sprite** media handler for this operation.

spriteIndex

The index of the **sprite** for this operation.

propertyType

The property whose value should be set (see below).

propertyValue

A pointer to a variable that contains the new value of the selected property. The type of data you pass for this parameter depends on the property type.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

propertyType Constants

`kSpritePropertyMatrix`

The *propertyValue* parameter returns the **sprite**’s `MatrixRecord` (IV–2304) structure.

`kSpritePropertyImageDescription`

The *propertyValue* parameter returns an `ImageDescriptionHandle`, which is a handle to the **sprite**’s `ImageDescription` (IV–2285) structure.

SpriteMediaSetSpriteProperty

kSpritePropertyImageDataPtr

The `propertyValue` parameter returns a `Ptr`, which is a pointer to the **sprite**'s image data.

kSpritePropertyVisible

The `propertyValue` parameter returns a Boolean that is `TRUE` if the **sprite** is visible, `FALSE` otherwise.

kSpritePropertyLayer

The `propertyValue` parameter returns the **sprite**'s layer number.

kSpritePropertyGraphicsMode

The `propertyValue` parameter returns the **sprite**'s `ModifierTrackGraphicsModeRecord` (IV–2311).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

See Also

For the corresponding get function, see `SpriteMediaGetProperty` (III–1893).

SpriteMediaSetSpriteProperty

Sets the specified property of a **sprite**.

```
ComponentResult SpriteMediaSetSpriteProperty (
    MediaHandler      mh,
    QTAtomID          spriteID,
    long               propertyType,
    void               *propertyValue );
```

`mh`

The **sprite** media handler for this operation.

`spriteIndex`

The index of the **sprite** for this operation.

`propertyType`

The property whose value should be retrieved (see below).

`propertyValue`

A pointer to the new value of the selected property. The type of data you pass for this parameter depends on the property type.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

propertyType Constants

kSpritePropertyMatrix

The *propertyValue* parameter returns the **sprite**’s MatrixRecord (IV–2304) structure.

kSpritePropertyImageDescription

The *propertyValue* parameter returns an ImageDescriptionHandle, which is a handle to the **sprite**’s ImageDescription (IV–2285) structure.

kSpritePropertyImageDataPtr

The *propertyValue* parameter returns a Ptr, which is a pointer to the **sprite**’s image data.

kSpritePropertyVisible

The *propertyValue* parameter returns a Boolean that is TRUE if the **sprite** is visible, FALSE otherwise.

kSpritePropertyLayer

The *propertyValue* parameter returns the **sprite**’s layer number.

kSpritePropertyGraphicsMode

The *propertyValue* parameter returns the **sprite**’s ModifierTrackGraphicsModeRecord (IV–2311).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Managing Movie Sprites” (V–2902)

Related Java Methods

```
quicktime.std.movies.media.SpriteMediaHandler.setMatrix(),
quicktime.std.movies.media.SpriteMediaHandler.setVisible(),
quicktime.std.movies.media.SpriteMediaHandler.setLayer(),
quicktime.std.movies.media.SpriteMediaHandler.setGraphicsMode(),
quicktime.std.movies.media.SpriteMediaHandler.setImageIndex()
```

See Also

For the corresponding get function, see SpriteMediaGetSpriteProperty (III–1896).

SpriteMediaSpriteIDToIndex

SpriteMediaSpriteIDToIndex

Converts a **sprite** ID to the corresponding sprite index.

```
ComponentResult SpriteMediaSpriteIDToIndex (  
    MediaHandler    mh,  
    QTAtomID        spriteID,  
    short           *spriteIndex );
```

mh

The **sprite** media handler for this operation.

spriteID

The ID of the **sprite** for this operation.

spriteIndex

On return, a pointer to the index of the **sprite**.

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Managing Movie Sprites” (V–2902)

Related Java Methods

quicktime.std.movies.media.SpriteMediaHandler.spriteIDtoIndex()

SpriteMediaSpriteIndexToID

Returns the ID of a **sprite** specified by a sprite index.

```
ComponentResult SpriteMediaSpriteIndexToID (  
    MediaHandler    mh,  
    short           spriteIndex,  
    QTAtomID        *spriteID );
```

mh

The **sprite** media handler for this operation.

spriteIndex

The index of the **sprite** for this operation.

`spriteID`

A pointer to the **sprite** ID corresponding to the sprite index. If a sprite with the specified index does not exist, the error `paramErr` is returned.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing Movie Sprites” (V-2902)

Related Java Methods

`quicktime.std.movies.media.SpriteMediaHandler.spriteIndexToID()`

SpriteWorldHitTest

Determines whether any **sprites** are at a specified location in a sprite world.

```
OSErr SpriteWorldHitTest (
    SpriteWorld    theSpriteWorld,
    long           flags,
    Point          loc,
    Sprite         *spriteHit );
```

`theSpriteWorld`

The **sprite** world for this operation.

`flags`

Specifies flags (see below) that control the hit testing operation.

`loc`

A point in the **sprite** world’s display space to test for the existence of a sprite.

`spriteHit`

A pointer to a field that is to receive a **sprite** identifier. On return, this field contains the identifier of the frontmost sprite at the location specified by the `loc` parameter. If no sprite exists at the location, the function sets the value of this parameter to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

SpriteWorldIdle

flags Constants

`spriteHitTestBounds`

The specified location must be within the **sprite**'s bounding box.

`spriteHitTestImage`

If both this flag and **spriteHitTestBounds** are set, the specified location must be within the shape of the **sprite**'s image.

`spriteHitTestInvisibleSprites`

This flag enables invisible **sprites** to be hit tested.

`spriteHitTestIsClick`

This flag is for codecs that want mouse events, such as the Ripple codec.

`spriteHitTestLocInDisplayCoordinates`

Set this flag if you want to pass a display coordinate point in the `loc` parameter.

Discussion

If you are drawing the **sprite** world in a window, you should convert the location to your window's local coordinate system before passing it to `SpriteWorldHitTest`. A hit testing operation does not occur unless you pass either `spriteHitTestBounds` or `spriteHitTestImage` in the `flags` parameter. You can add other flags as needed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: "Working with Sprite Worlds" (V-2900)

Related Java Methods

`quicktime.std.anim.SpriteWorld.hitTest()`

SpriteWorldIdle

Allows a **sprite** world to update its invalid areas.

```
OSErr SpriteWorldIdle (
    SpriteWorld    theSpriteWorld,
    long           flagsIn,
    long           *flagsOut );
```

`theSpriteWorld`

The **sprite** world for this operation.

`flagsIn`

Contains flags (see below) describing actions that may take place during the idle. For the default behavior, set this parameter to 0.

`flagsOut`

On return, a pointer to flags (see below) describing actions that took place during the idle period. This parameter is optional; if you do not need the information, set it to NIL.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

flagsIn Constants

`kOnlyDrawToSpriteWorld`

Set this flag to indicate that drawing should take place in the **sprite** world only; drawing to the final destination should be suppressed.

`kSpriteWorldPreFlight`

Set this flag to determine whether the **sprite** world has any invalid areas that need to be drawn. If so, `SpriteWorldIdle` returns the `kSpriteWorldNeedsToDraw` flag in the `flagsOut` parameter.

flagsOut Constants

`kSpriteWorldDidDraw`

If set, this flag indicates that `SpriteWorldIdle` updated the **sprite** world.

`kSpriteWorldNeedsToDraw`

If set, this flag indicates that the **sprite** world has invalid areas that need to be drawn.

Discussion

This is the only **sprite** function that causes drawing to occur; you should call it as often as is necessary. Typically, you would make changes in perspective for a number of sprites and then call `SpriteWorldIdle` to redraw the changed sprites.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working with Sprite Worlds” (V-2900)

Related Java Methods

`quicktime.std.anim.SpriteWorld.idle()`

StandardGetFilePreview

Displays file **previews** in an Open dialog box using a standard file reply structure.

```
void StandardGetFilePreview (
    FileFilterUPP      fileFilter,
    short              numTypes,
    ConstSFTYPEListPtr typeList,
    StandardFileReply   *reply );
```

fileFilter

Points to a FileFilterProc (III–2082) callback that filters the files that are displayed to the user in the dialog box. This is an optional function provided by your application; if you don’t want to supply a filter function, set this parameter to NIL. StandardGetFilePreview uses this parameter along with the numTypes and typeList parameters to determine which files appear in the dialog box.

numTypes

The number of file types in the array specified by the typeList parameter (a number between 1 and 4). Set this parameter to –1 to display all files.

typeList

Specifies an array of file types to be displayed to the user. This function only displays files whose type matches an entry in this array (unless you set the numTypes parameter to –1; in this case, the function displays all files to the user).

reply

A pointer to the StandardFileReply (IV–2461) structure that is to receive information about the user’s selection.

Discussion

This function automatically converts files to movies if your application requests movies. If a file could be converted into a movie file using a movie import component, then the file is shown in the Standard File dialog box. The following code illustrates a typical use of this function:

```
// StandardGetFilePreview coding example
// See “Discovering QuickTime,” page 385
Movie MyGetMovie (void)
{
    OSErr          nErr;
    SFTYPEList      types = {MovieFileType, 0, 0, 0};
    StandardFileReply sfr;
    Movie           movie = NIL;
    short           nFileRefNum;
```

```

StandardGetFilePreview(NIL, 1, types, &sfr);
if (sfr.sfGood) {
    nErr = OpenMovieFile(&sfr.sfFile, &nFileRefNum, fsRdPerm);
    if (nErr == noErr) {
        short          nResID = 0;           //We want the first movie.
        Str255          strName;
        Boolean          bWasChanged;

        nErr = NewMovieFromFile(&movie, nFileRefNum, &nResID, strName,
                                newMovieActive, &bWasChanged);
        CloseMovieFile(nFileRefNum);
    }
}
return movie;
}

```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Displaying File Previews” (V–2758)

Related Java Methods

quicktime.io.QTFile.standardGetFilePreview()

StartMovie

Starts the movie playing from the current movie time.

```

void StartMovie (
    Movie    theMovie );

```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

StdPix

Discussion

You are not required to call this function to start a movie. It is included in the QuickTime API for convenience. Before playing the movie, the Movie Toolbox makes the movie active, prerolls the movie, and sets the movie to its preferred playback rate. You can use `SetMoviePreferredRate` (III–1626) to change this setting.

Special Considerations

A movie’s current time is saved when a movie is stored in a movie file. Therefore, your application should appropriately position a movie before playing the movie. Use `GoToBeginningOfMovie` (I–550) to set a movie to play from its start.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Controlling Movie Playback” (V–2717)

Related Java Methods

`quicktime.std.movies.Movie.start()`

StdPix

Extends the `grafProcs` field of the `CGrafPort` (IV–2168) structure to support compressed data, mattes, matrices, and pixel maps, letting you intercept image data in compressed form before it is decompressed and displayed.

```
void StdPix (
    PixMapPtr      src,
    const Rect      *srcRect,
    MatrixRecordPtr matrix,
    short           mode,
    RgnHandle       mask,
    PixMapPtr       matte,
    const Rect      *matteRect,
    short           flags );
```

`src`

Contains a pointer to a `PixMap` (IV–2332) structure containing the image to draw. Use `GetCompressedPixMapInfo` (I–397) to retrieve information about this structure.

`srcRect`

Points to a `Rect` (IV–2399) structure that defines the portion of the image to display. This rectangle must lie within the boundary rectangle of the

compressed image or within the source image. If this parameter is set to `NIL`, the entire image is displayed.

`matrix`

Contains a pointer to a `MatrixRecord` (IV–2304) structure that specifies the mapping of the source rectangle to the destination. It is a fixed-point, 3-by-3 matrix.

`mode`

Specifies the transfer mode for the operation; see “Graphics Transfer Modes” (IV–2670). Note that this parameter also controls the accuracy of any decompression operation that may be required to display the image. If bit 7 (0x80) of the `mode` parameter is set to 1, the `StdPix` function sets the decompression accuracy to `codecNormalQuality`. If this bit is set to 0, the function sets the accuracy to `codecHighQuality`.

`mask`

Contains a handle to a clipping region in the destination coordinate system. If specified, the compressor applies this mask to the destination image. If there is no mask, this parameter is set to `NIL`.

`matte`

Points to a `PixMap` (IV–2332) structure that contains a blend matte. The blend matte causes the decompressed image to be blended into the destination pixel map. The matte can be defined at any supported pixel depth; the matte depth need not correspond to the source or destination depths. However, the matte must be in the coordinate system of the source image. If there is no matte, this parameter is set to `NIL`.

`matteRect`

Contains a pointer to a `Rect` (IV–2399) structure that defines a portion of the blend matte to apply. This parameter is set to `NIL` if there is no matte or if the entire matte is to be used.

`flags`

Contains control flags (see below).

flags Constants

`call101dBits`

If this flag is set, the function calls `QuickDraw`’s `bitsProc` routine with the decompressed image data. A pointer to this routine is located in the `bitsProc` field of the `CQDProcs` (IV–2226) structure. If the `bitsProc` routine is not customized, it is not called unless the `callStdBits` flag is also set.

StopMovie

`callStdBits`

If this flag is set, the `callOldBits` flag is set, and the `CQDProcs` (IV–2226) structure’s `bitsProc` field is set to the Mac OS `StdBits` routine. Then the `StdBits` routine is called with the decompressed image data.

`noDefaultOpcodes`

If this flag is set and a `Picture` (IV–2331) structure is open for writing, the default picture opcodes (for displaying a warning when QuickTime is not installed) are not added to the output structure. This can be useful when storing multiple `StdPix` opcodes in a single structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With the `StdPix` Function” (V–2797)

StopMovie

Stops the playback of a movie.

```
void StopMovie (
    Movie    theMovie );
```

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result You can access this function’s error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Controlling Movie Playback” (V–2717)

Related Java Methods

`quicktime.std.movies.Movie.stop()`

SubtractTime

Subtracts one time from another.

```
void SubtractTime (
    TimeRecord      *dst,
    const TimeRecord *src );
```

dst

A pointer to a TimeRecord (IV–2486) structure. This **time structure** contains one of the operands for the subtraction. This function returns the result of the subtraction into this time structure as a duration.

src

A pointer to a TimeRecord (IV–2486) structure. The Movie Toolbox subtracts this value from the time or duration specified by the *dst* parameter.

function result You can access error returns from this function through GetMoviesError (I–492) and GetMoviesStickyError (I–494). See “Error Codes” (IV–2663).

Discussion

If the two times are relative to different time scales or time bases, this function converts the times as appropriate to yield reasonable results. However, the time bases for both time values must rely on the same time source.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Movies.h

Programming summary: “Working With Times” (V–2762)

Related Java Methods

```
quicktime.std.clocks.TimeRecord.subtractTime()
```

SysBeep

Plays the system alert sound.

```
void SysBeep (
    short    duration );
```

TCFrameNumberToTimeCode

duration

The duration in sixtieths of a second (ticks) of the resulting sound. This parameter is ignored except on a Macintosh Plus, Macintosh SE, or Macintosh Classic when the system alert sound is the Simple Beep. The recommended duration is 30 **ticks**, which equals one-half second.

function result You can access error returns from this function through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494). See “Error Codes” (IV–2663).

Discussion

This function causes the Sound Manager to play the system alert sound at its current volume. If necessary, the Sound Manager loads into memory the sound **resource** containing the system alert sound and links it to a sound channel. The Macintosh user selects a system alert sound in the Alert Sounds subpanel of the Sound control panel. The volume of the sound produced depends on the current setting of the system alert sound volume, which the user can adjust in the Alert Sounds subpanel of the Sound control panel. The system alert sound volume can also be read and set by calling `GetSysBeepVolume` (I–514) and `SetSysBeepVolume` (III–1647). If the volume is set to 0 (silent) and the system alert sound is enabled, calling `SysBeep` causes the menu bar to blink once.

Special Considerations

Because this function moves memory, you should not call it at interrupt time

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Sound.h`

T

TCFrameNumberToTimeCode

Converts a frame number into its corresponding timecode time value.

```
HandlerError TCFrameNumberToTimeCode (  
    MediaHandler      mh,  
    long              frameNumber,  
    TimeCodeDef       *tcdef,
```

```
TimeCodeRecord    *tcrec );
```

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (I–432).

frameNumber

The frame number that is to be converted.

tcdef

A pointer to the `TimeCodeDef` (IV–2482) structure to use for the conversion.

tcrec

A pointer to the `TimeCodeRecord` (IV–2484) structure that is to receive the time value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With the Timecode Media Handler” (V–2870)

Related Java Methods

`quicktime.std.qtcomponents.TimeCoder.toTimeCode()`

TCGetCurrentTimeCode

Retrieves the timecode and source identification information for the current movie time.

```
HandlerError TCGetCurrentTimeCode (
    MediaHandler    mh,
    long            *frameNum,
    TimeCodeDef     *tcdef,
    TimeCodeRecord  *tcrec,
    UserData        *srcRefH );
```

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (I–432).

TCGetDisplayOptions

frameNum

A pointer to a field that is to receive the current frame number. Set this field to NIL if you don't want to retrieve the frame number.

tcdef

A pointer to a TimeCodeDef (IV-2482) structure. The media handler returns the movie's timecode definition information. Set this parameter to NIL if you don't want this information.

tcrec

A pointer to a TimeCodeRecord (IV-2484) structure. The media handler returns the current time value. Set this parameter to NIL if you don't want this information.

srcRefH

A pointer to a field that is to receive a handle containing the source information as a UserDataRecord (IV-2496) structure. It is your responsibility to dispose of this structure when you are done with it. Set this field to NIL if you don't want this information.

function result See "Error Codes" (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: "Working With the Timecode Media Handler" (V-2870)

Related Java Methods

```
quicktime.util.QTHandle.fromTimeCoderCurrent(),  
quicktime.std.qtcomponents.TimeCoder.getCurrent()
```

TCGetDisplayOptions

Retrieves the text characteristics that apply to timecode information displayed in a movie.

```
HandlerError TCGetDisplayOptions (  
    MediaHandler      mh,  
    TCTextOptionsPtr  textOptions );
```

mh

The timecode media handler. You obtain this identifier by calling GetMediaHandler (I-432).

`textOptions`

A pointer to a `TCTextOptions` (IV–2468) structure. This structure will receive font and style information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With the Timecode Media Handler” (V–2870)

Related Java Methods

`quicktime.std.qtcomponents.TimeCoder.getDisplayOptions()`

See Also

For the corresponding set function, see `TCSetDisplayOptions` (III–1922).

TCGetSourceRef

Retrieves the source information from the timecode media sample reference.

```
HandlerError TCGetSourceRef (
    MediaHandler          mh,
    TimeCodeDescriptionHandle tcdH,
    UserData              *srefH );
```

`mh`

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (I–432).

`tcdH`

Specifies a handle to a `TimeCodeDescription` (IV–2483) structure that defines the media sample reference for this operation.

`srefH`

Specifies a pointer to a handle that will receive the source information as a `UserDataRecord` (IV–2496) structure. It is your application’s responsibility to dispose of this structure when you are done with it.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

TCGetTimeCodeAtTime

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With the Timecode Media Handler” (V–2870)

Related Java Methods

```
quicktime.util.QTHandle.fromTimeCoderSource(),  
quicktime.std.qtcomponents.TimeCoder.getSourceRef()
```

See Also

For the corresponding set function, see `TCSetSourceRef` (III–1922).

TCGetTimeCodeAtTime

Returns a track’s timecode information corresponding to a specific media time.

```
HandlerError TCGetTimeCodeAtTime (  
    MediaHandler      mh,  
    TimeValue         mediaTime,  
    long              *frameNum,  
    TimeCodeDef        *tcdef,  
    TimeCodeRecord     *tcdata,  
    UserData           *srcRefH );
```

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (I–432).

mediaTime

A time value for which you want to retrieve timecode information. This time value is expressed in the media’s time coordinate system.

frameNum

A pointer to a field that is to receive the current frame number. Set this field to `NIL` if you don’t want to retrieve the frame number.

tcdef

A pointer to a `TimeCodeDef` (IV–2482) structure. The media handler returns the movie’s timecode definition information. Set this parameter to `NIL` if you don’t want this information.

tcdata

A pointer to a `TimeCodeRecord` (IV–2484) structure. The media handler returns the current time value. Set this parameter to `NIL` if you don’t want this information.

srcRefH

A pointer to a field that is to receive a handle containing the source information as a `UserDataRecord` (IV–2496) structure. It is your responsibility to dispose of this structure when you are done with it. Set this field to `NIL` if you don’t want this information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With the Timecode Media Handler” (V–2870)

Related Java Methods

`quicktime.util.QTHandle.fromTimeCoderTime()`,
`quicktime.std.qtcomponents.TimeCoder.getAtTime()`

TCGetTimeCodeFlags

Retrieves the timecode control flags.

```
HandlerError TCGetTimeCodeFlags (
    MediaHandler    mh,
    long            *flags );
```

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (I–432).

flags

A pointer to a field that is to receive a control flag (see below).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags Constant

`tcdfShowTimeCode`

Controls the display of timecode information. If this flag is set to 1, the timecode information is displayed when the movie is played. The timecode track must be enabled in order for the timecode information to be displayed.

Version Notes

Introduced in QuickTime 3 or earlier.

TCSetDisplayOptions

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With the Timecode Media Handler” (V–2870)

Related Java Methods

`quicktime.std.qtcomponents.TimeCoder.getFlags()`

See Also

For the corresponding set function, see `TCSetTimeCodeFlags` (III–1923).

TCSetDisplayOptions

Sets the text characteristics that apply to timecode information displayed in a movie.

```
HandlerError TCSetDisplayOptions (  
    MediaHandler      mh,  
    TCTextOptionsPtr  textOptions );
```

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (I–432).

textOptions

A pointer to a `TCTextOptions` (IV–2468) structure. This structure contains font and style information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With the Timecode Media Handler” (V–2870)

Related Java Methods

`quicktime.std.qtcomponents.TimeCoder.setDisplayOptions()`

See Also

For the corresponding get function, see `TCGetDisplayOptions` (III–1918).

TCSetSourceRef

Changes the source information in the timecode media sample reference.


```
HandlerError TCSetSourceRef (
    MediaHandler      mh,
    TimeCodeDescriptionHandle tcdH,
    UserData          srefH );
```

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (I–432).

tcdH

Specifies a handle containing the timecode media sample reference that is to be updated.

srefH

Specifies a handle to the source information to be placed in the sample reference as a `UserDataRecord` (IV–2496) structure. It is your application’s responsibility to dispose of this structure when you are done with it.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With the Timecode Media Handler” (V–2870)

Related Java Methods

`quicktime.std.qtcomponents.TimeCoder.setSourceRef()`

See Also

For the corresponding get function, see `TCGetSourceRef` (III–1919).

TCSetTimeCodeFlags

Changes the flag that affects how the toolbox handles timecode information.

```
HandlerError TCSetTimeCodeFlags (
    MediaHandler      mh,
    long              flags,
    long              flagsMask );
```

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (I–432).

TCTimeCodeToFrameNumber

`flags`

The new flag value.

`flagsMask`

Specifies which of the flag values are to change. The media handler modifies only those flag values that correspond to bits that are set to 1 in this parameter. Use the flag values from the `flags` parameter. To turn off timecode display, set the `tcdfShowTimeCode` flag to 1 in the `flagsMask` parameter and to 0 in the `flags` parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

flags and flagsMask Constant

`tcdfShowTimeCode`

Controls the display of timecode information. If this flag is set to 1, the timecode information is displayed when the movie is played. The timecode track must be enabled in order for the timecode information to be displayed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With the Timecode Media Handler” (V–2870)

Related Java Methods

`quicktime.std.qtcomponents.TimeCoder.setFlags()`

See Also

For the corresponding get function, see `TCGetTimeCodeFlags` (III–1921).

TCTimeCodeToFrameNumber

Converts a timecode time value into its corresponding frame number.

```
HandlerError TCTimeCodeToFrameNumber (
    MediaHandler      mh,
    TimeCodeDef       *tcdef,
    TimeCodeRecord     *tcrec,
    long               *frameNumber );
```

`mh`

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (I–432).

tcdef

A pointer to the TimeCodeDef (IV–2482) structure to use for the conversion.

tcrec

A pointer to the TimeCodeRecord (IV–2484) structure containing the time value to convert.

frameNumber

A pointer to a field that is to receive the frame number that corresponds to the time value in the tcrec parameter.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Working With the Timecode Media Handler” (V–2870)

Related Java Methods

quicktime.std.qtcomponents.TimeCoder.toFrameNumber()

TCTimeCodeToString

Converts a time value into a text string (HH:MM:SS:FF).

```
HandlerError TCTimeCodeToString (
    MediaHandler      mh,
    TimeCodeDef       *tcdef,
    TimeCodeRecord     *tcrec,
    StringPtr         tcStr );
```

mh

The timecode media handler. You obtain this identifier by calling GetMediaHandler (I–432).

tcdef

A pointer to the TimeCodeDef (IV–2482) structure to use for the conversion.

tcrec

A pointer to the TimeCodeRecord (IV–2484) structure to use for the conversion.

tcStr

A pointer to a text string that is to receive the converted time value.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

TerminateQHdr

Discussion

If the timecode uses the dropframe technique, the separators are semicolons (;) rather than colons (:).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Working With the Timecode Media Handler” (V–2870)

Related Java Methods

`quicktime.std.qtcomponents.TimeCoder.timeCodeToString()`

TerminateQHdr

Terminates a Windows QHdr data structure.

```
void TerminateQHdr (  
    QHdr      *qhdr );
```

```
qhdr  
    qhdr
```

Discussion

The `TerminateQHdr` function deallocates the data structures created by `InitializeQHdr` (II–756).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QTML.h`

TerminateQTML

Terminates the QuickTime Media Layer.

```
void TerminateQTML ( void );
```

Discussion

Use `TerminateQTML` to terminate a QTML session after calling `ExitMovies` (I–340). You should not make this call from a QuickTime component, such as an image decompressor; it is provided only for host applications.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

Related Java Methods

quicktime.QTSession.terminate()

TerminateQTS

Terminates the QuickTime Streaming toolbox.

```
OSErr TerminateQTS ( void );
```

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: QuickTimeStreaming.h

TerminateQTVR

Terminates QuickTime VR when you have finished using its functions.

```
OSErr TerminateQTVR ( void );
```

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Discussion

You must use the `TerminateQTVR` function when you have finished using the functions of QuickTime VR. Multiple calls to `InitializeQTVR` (II-758) and `TerminateQTVR` can be either nested or sequential, but there must be at least one call to `TerminateQTVR` corresponding to each call to `InitializeQTVR`.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeVR.h

Programming summary: “Initializing and Terminating QuickTime VR” (V-2904)

TextExportGetDisplayData

Related Java Methods

`quicktime.QTSession.terminateVR()`

TextExportGetDisplayData

Retrieves text display information for the current sample in the specified text export component.

```
ComponentResult TextExportGetDisplayData (  
    TextExportComponent    ci,  
    TextDisplayData        *textDisplay );
```

ci

Specifies the text export component for this operation. Applications can obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

textDisplay

Contains a pointer to a `TextDisplayData` (IV–2476) structure. On return, this structure contains the display settings of the current text sample.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You call this function to retrieve the text display data structure for a text sample. The text display data structure contains the formatting information for the text sample. When the text export component exports a text sample, it uses the information in this structure to generate the appropriate text descriptors for the sample. Likewise, when the text import component imports a text sample, it sets the appropriate fields in the text display data structure based on the sample’s text descriptors.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Text” (V–2860)

TextExportGetSettings

Retrieves the value of the text export option for the specified text export component.

```
ComponentResult TextExportGetSettings (
```

```
TextExportComponent    ci,
long                  *setting );
```

ci

Specifies the text export component for this operation. Applications can obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

setting

Contains a pointer to a 32-bit integer. On return, this integer contains a constant (see below) that represents the current value of the text export option.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

setting Constants

`kMovieExportTextOnly`

The text export component exports text without time descriptors or time stamps.

`kMovieExportAbsoluteTime`

The text export component exports text along with its text descriptors and time stamps. For each sample, calculate the time stamp relative to the start of the movie.

`kMovieExportRelativeTime`

The text export component exports text along with its text descriptors and time stamps. For each sample, calculate the time stamp relative to the previous time stamp.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Text” (V–2860)

See Also

For the corresponding set function, see `TextExportSetSettings` (III–1930).

TextExportGetTimeFraction

Retrieves the time scale the specified text export component uses to calculate time stamps.

```
ComponentResult TextExportGetTimeFraction (
    TextExportComponent    ci,
    long                  *movieTimeFraction );
```

TextExportSetSettings

ci

Specifies the text export component for this operation. Applications can obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

movieTimeFraction

Contains a pointer to a 32-bit integer. On return, this integer contains the time scale used in the fractional part of time stamps.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

You call this function to retrieve the time scale used by the text export component to calculate the fractional part of time stamps. You set a text component’s time scale by specifying it in the Text Export Settings dialog box or by calling `TextExportSetTimeFraction` (III–1931).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Text” (V–2860)

See Also

For the corresponding set function, see `TextExportSetTimeFraction` (III–1931).

TextExportSetSettings

Sets the value of the text export option for the specified text export component.

```
ComponentResult TextExportSetSettings (
    TextExportComponent  ci,
    long                 setting );
```

ci

Specifies the text export component for this operation. Applications can obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

setting

A constant (see below) that specifies the new value of the text export option.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

setting Constants

`kMovieExportTextOnly`

The text export component exports text without time descriptors or time stamps.

`kMovieExportAbsoluteTime`

The text export component exports text along with its text descriptors and time stamps. For each sample, calculate the time stamp relative to the start of the movie.

`kMovieExportRelativeTime`

The text export component exports text along with its text descriptors and time stamps. For each sample, calculate the time stamp relative to the previous time stamp.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Text” (V–2860)

See Also

For the corresponding get function, see `TextExportGetSettings` (III–1928).

TextExportSetTimeFraction

Sets the time scale the specified text export component uses to calculate time stamps.

```
ComponentResult TextExportSetTimeFraction (
    TextExportComponent  ci,
    long                 movieTimeFraction );
```

ci

Specifies the text export component for this operation. Applications can obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

movieTimeFraction

Specifies the time scale used in the fractional part of time stamps. The value should be between 1 and 10000, inclusive.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

TextMediaAddHiliteSample

Discussion

You call this function to set the time scale used by the text export component to calculate the fractional part of time stamps. You can also set a text component's time scale by specifying it in the text export settings dialog box. You can retrieve a text component's time scale by calling `TextExportGetTimeFraction` (III–1929).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Exporting Text” (V–2860)

See Also

For the corresponding get function, see `TextExportGetTimeFraction` (III–1929).

TextMediaAddHiliteSample

Provides dynamic highlighting of text.

```
ComponentResult TextMediaAddHiliteSample (
    MediaHandler      mh,
    short             hiliteStart,
    short             hiliteEnd,
    RGBColor          *rgbHiliteColor,
    TimeValue         duration,
    TimeValue         *sampleTime );
```

`mh`

The media handler for the text media obtained by `GetMediaHandler` (I–432).

`hiliteStart`

Indicates the beginning of the text to be highlighted.

`hiliteEnd`

Indicates the ending of the text to be highlighted. If the value of the `hiliteStart` parameter equals that of the `hiliteEnd` parameter, then no text is highlighted (that is, highlighting is turned off for the duration of the specified sample).

`rgbHiliteColor`

A pointer to the `RGBColor` (IV–2403) structure that defines the color for highlighting. If this parameter is not `NIL`, then the specified color is used when highlighting the text indicated by the `hiliteStart` and `hiliteEnd` parameters. Otherwise, the default system highlight color is used.

duration

Specifies how long the text sample should last. This duration is expressed in the media's time base.

sampleTime

A pointer to a time value. The actual media time at which the sample was added is returned here.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Text Media Handler Functions” (V-2756)

TextMediaAddTESample

Specifies a `TextEdit` handle to be added to a specified media.

```
ComponentResult TextMediaAddTESample (
    MediaHandler      mh,
    TEHandle          hTE,
    RGBColor          *backColor,
    short              textJustification,
    Rect              *textBox,
    long              displayFlags,
    TimeValue          scrollDelay,
    short              hiliteStart,
    short              hiliteEnd,
    RGBColor          *rgbHiliteColor,
    TimeValue          duration,
    TimeValue          *sampleTime );
```

mh

The media handler for the text media obtained by `GetMediaHandler` (I-432).

hTE

A handle to a `TERec` (IV-2469) structure.

backColor

A pointer to an `RGBColor` (IV-2403) structure specifying the text background color. Passing `NIL` for this parameter in defaults to white.

TextMediaAddTESample

`textJustification`

Indicates the justification of the text (see below).

`textBox`

A pointer to a `Rect` (IV–2399) structure that defines the box within which the text is to be displayed. The box is relative to the track bounds.

`displayFlags`

Contains the text display flags (see below).

`scrollDelay`

Indicates the delay in scrolling associated with the setting of the `dfScrollIn` and `dfScrollOut` display flags. If the value of the `scrollDelay` parameter is greater than 0 and the `dfScrollIn` flag is set, the text pauses when it has scrolled all the way in for the amount of time specified by `scrollDelay`. If the `dfScrollOut` flag is set, the pause occurs first before the text scrolls out. If both these flags are set, the pause occurs at the midpoint between scrolling in and scrolling out.

`hiliteStart`

The beginning of the text to be highlighted.

`hiliteEnd`

The end of the text to be highlighted. If the `hiliteEnd` parameter is greater than the `hiliteStart` parameter, then the text is highlighted from the selection specified by `hiliteStart` to `hiliteEnd`. To specify additional highlighting, you can use `TextMediaAddHiliteSample` (III–1932).

`rgbHiliteColor`

Contains a pointer to an `RGBColor` (IV–2403) structure that defines the color for highlighting. If this parameter is not `NIL`, then the specified color is used when highlighting the text indicated by the `hiliteStart` and `hiliteEnd` parameters. Otherwise, the default system highlight color is used.

`duration`

A time value that specifies how long the text sample should last. This duration is expressed in the media's time base.

`sampleTime`

Contains a pointer to a time value. The actual media time at which the sample was added is returned here.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

textJustification Constants

teCenter

Text is centered.

teFlushRight

Text is lined up on the right.

teFlushLeft

Text is lined up on the left.

teFlushDefault

Text is lined up left or right, depending on the script being used.

displayFlags Constants

dfDontDisplay

Does not display the specified sample.

dfDontAutoScale

Does not scale the text if the track bounds increase.

dfClipToTextBox

Clips to the text box only. This is useful if the text overlays the video.

dfShrinkTextBoxToFit

Recalculates size of the `textBox` parameter to just fit the given text and stores this rectangle with the text data.

dfScrollIn

Scrolls the text in until the last of the text is in view.

dfScrollOut

Scrolls text out until the last of the text is out of view. If both `dfScrollIn` and `dfScrollOut` are set, the text is scrolled in, then out.

dfHorizScroll

Scrolls a single line of text horizontally. If the `dfHorizScroll` flag is not set, then the scrolling is vertical.

dfReverseScroll

If set, scrolls vertically down, rather than up. If not set, horizontal scrolling proceeds toward the left rather than toward the right.

dfContinuousScroll

If this flag is set, the text media handler lets new samples cause previous samples to scroll out. You must also set `dfScrollIn` or `dfScrollOut`, or both, for this to take effect.

dfFlowHoriz

If this flag is set, the text media handler lets horizontally scrolled text flow within the text box instead of extending to the right.

TextMediaAddTextSample

dfContinuousKaraoke

If this flag is set, the text media handler ignores the starting offset when highlighting text. Instead, it highlights text from the beginning of the text sample to the ending offset.

dfDropShadow

If this flag is set, the text media handler displays text with a drop shadow. If you use `TextMediaSetTextSampleData` (III–1950), the position and translucency of the drop shadow is under your application’s control.

dfAntiAlias

If this flag is set, the text media handler displays text with anti-aliasing. Note that although anti-aliased text looks smoother, anti-aliasing can slow down performance.

dfKeyedText

If this flag is set, the text media handler renders text over the background without drawing the background color. This technique is also known as “masked text.”

dfInverseHilite

If this flag is set, the text media handler highlights text using inverse video instead of the highlight color.

dfTextColorHilite

If this flag is set, the text media handler highlights text by changing the color of the text.

Special Considerations

Be sure to turn on the `dfDropShadow` display flag after you call this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Text Media Handler Functions” (V–2756)

TextMediaAddTextSample

Adds a single block of styled text to an existing media.

```
ComponentResult TextMediaAddTextSample (
    MediaHandler    mh,
    Ptr             text,
    unsigned long    size,
```

```

short      fontNumber,
short      fontSize,
Style      textFace,
RGBColor   *textColor,
RGBColor   *backColor,
short      textJustification,
Rect       *textBox,
long       displayFlags,
TimeValue  scrollDelay,
short      hiliteStart,
short      hiliteEnd,
RGBColor   *rgbHiliteColor,
TimeValue  duration,
TimeValue  *sampleTime );

```

mh

The media handler for the text media obtained by GetMediaHandler (I-432).

text

A pointer to a block of text.

size

Indicates the size of the text block, in bytes.

fontNumber

The number for the font in which to display the text.

fontSize

Indicates the size of the font.

textFace

Indicates the typeface or style of the text (that is, bold, italic, and so on).

textColor

A pointer to an RGBColor (IV-2403) structure specifying the color of the text. Passing NIL for this parameter in defaults to black.

backColor

A pointer to an RGBColor (IV-2403) structure specifying the text background color. Passing NIL for this parameter in defaults to white.

textJustification

Indicates the justification of the text (see below).

textBox

A pointer to a Rect (IV-2399) structure that defines the box within which the text is to be displayed. The box is relative to the track bounds.

displayFlags

Contains the text display flags (see below).

T

TextMediaAddTextSample

`scrollDelay`

Indicates the delay in scrolling associated with the setting of the `dfScrollIn` and `dfScrollOut` display flags. If the value of the `scrollDelay` parameter is greater than 0 and the `dfScrollIn` flag is set, the text pauses when it has scrolled all the way in for the amount of time specified by `scrollDelay`. If the `dfScrollOut` flag is set, the pause occurs first before the text scrolls out. If both these flags are set, the pause occurs at the midpoint between scrolling in and scrolling out.

`hiliteStart`

The beginning of the text to be highlighted.

`hiliteEnd`

The end of the text to be highlighted. If the `hiliteEnd` parameter is greater than the `hiliteStart` parameter, then the text is highlighted from the selection specified by `hiliteStart` to `hiliteEnd`. To specify additional highlighting, you can use `TextMediaAddHiliteSample` (III–1932).

`rgbHiliteColor`

Contains a pointer to an `RGBColor` (IV–2403) structure that defines the color for highlighting. If this parameter is not `NIL`, then the specified color is used when highlighting the text indicated by the `hiliteStart` and `hiliteEnd` parameters. Otherwise, the default system highlight color is used.

`duration`

A time value that specifies how long the text sample should last. This duration is expressed in the media’s time base.

`sampleTime`

Contains a pointer to a time value. The actual media time at which the sample was added is returned here.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

textJustification Constants

`teCenter`

Text is centered.

`teFlushRight`

Text is lined up on the right.

`teFlushLeft`

Text is lined up on the left.

`teFlushDefault`

Text is lined up left or right, depending on the script being used.

displayFlags Constants

`dfDontDisplay`

Does not display the specified sample.

`dfDontAutoScale`

Does not scale the text if the track bounds increase.

`dfClipToTextBox`

Clips to the text box only. This is useful if the text overlays the video.

`dfShrinkTextBoxToFit`

Recalculates size of the `textBox` parameter to just fit the given text and stores this rectangle with the text data.

`dfScrollIn`

Scrolls the text in until the last of the text is in view.

`dfScrollOut`

Scrolls text out until the last of the text is out of view. If both `dfScrollIn` and `dfScrollOut` are set, the text is scrolled in, then out.

`dfHorizScroll`

Scrolls a single line of text horizontally. If the `dfHorizScroll` flag is not set, then the scrolling is vertical.

`dfReverseScroll`

If set, scrolls vertically down, rather than up. If not set, horizontal scrolling proceeds toward the left rather than toward the right.

`dfContinuousScroll`

If this flag is set, the text media handler lets new samples cause previous samples to scroll out. You must also set `dfScrollIn` or `dfScrollOut`, or both, for this to take effect.

`dfFlowHoriz`

If this flag is set, the text media handler lets horizontally scrolled text flow within the text box instead of extending to the right.

`dfContinuousKaraoke`

If this flag is set, the text media handler ignores the starting offset when highlighting text. Instead, it highlights text from the beginning of the text sample to the ending offset.

`dfDropShadow`

If this flag is set, the text media handler displays text with a drop shadow. If you use `TextMediaSetTextSampleData` (III–1950), the position and translucency of the drop shadow is under your application's control.

TextMediaDrawRaw

dfAntiAlias

If this flag is set, the text media handler displays text with anti-aliasing. Note that although anti-aliased text looks smoother, anti-aliasing can slow down performance.

dfKeyedText

If this flag is set, the text media handler renders text over the background without drawing the background color. This technique is also known as “masked text.”

dfInverseHilite

If this flag is set, the text media handler highlights text using inverse video instead of the highlight color.

dfTextColorHilite

If this flag is set, the text media handler highlights text by changing the color of the text.

Special Considerations

Be sure to turn on the `dfDropShadow` display flag after you call this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Text Media Handler Functions” (V-2756)

Related Java Methods

`quicktime.std.movies.media.TextMediaHandler.addTextSample()`

TextMediaDrawRaw

Undocumented

```
ComponentResult TextMediaDrawRaw (
    MediaHandler      mh,
    GWorldPtr         gw,
    GDHandle          gd,
    void              *data,
    long              dataSize,
    TextDescriptionHandle tdh );
```

`mh`

The text media handler obtained by `GetMediaHandler` (I-432).

gw

A pointer to a `CGrafPort` (IV–2168) structure that defines a graphics world.

gd

A handle to a graphics device.

data

A pointer to the source data.

dataSize

The size of the source data.

tdh

A handle to a `TextDescription` (IV–2474) structure.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming InfoC interface file: `Movies.h`**TextMediaFindNextText**

Searches for text with a specified media handler starting at a given time.

```
ComponentResult TextMediaFindNextText (
    MediaHandler    mh,
    Ptr             text,
    long            size,
    short           findFlags,
    TimeValue       startTime,
    TimeValue       *foundTime,
    TimeValue       *foundDuration,
    long            *offset );
```

mh

The media handler for the text media obtained by `GetMediaHandler` (I–432).

text

Points to the text to be found.

size

The length of the text to be found.

TextMediaFindNextText

`findFlags`

Flags (see below) that determine the conditions of the search.

`startTime`

Indicates the time (expressed in the movie time scale) at which to begin the search.

`foundTime`

A pointer to the movie time at which the text sample is found if the search is successful. Otherwise, it returns `-1`.

`foundDuration`

A pointer to the duration of the sample (in the movie time scale) that is found if the search is successful.

`offset`

A pointer to the offset of the found text from the beginning of the text portion of the sample.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

findFlags Constants

`findTextEdgeOK`

Okay to find text at exactly the specified sample time.

`findTextCaseSensitive`

Case-sensitive search.

`findTextReverseSearch`

Search from start time backwards.

`findTextWrapAround`

Wrap search when beginning or end of movie is hit.

`findTextUseOffset`

Begin search at the given character offset into sample rather than edge.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Text Media Handler Functions” (V-2756)

Related Java Methods

`quicktime.std.movies.media.TextMediaHandler.findNextText()`

TextMediaGetTextProperty

Sets properties of a text media.

```
ComponentResult TextMediaGetTextProperty (
    MediaHandler      mh,
    TimeValue         atMediaTime,
    long              propertyType,
    void              *data,
    long              dataSize );
```

mh

The text media handler obtained by GetMediaHandler (I-432).

atMediaTime

The media time of the text.

propertyType

A constant (see below) that identifies the text property to be set.

data

A pointer to text data.

dataSize

The size of the data.

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

propertyType Constants

kTextTextHandle

A handle to the text.

kTextTextPtr

A pointer to the text.

kTextTEStyle

The text style.

kTextSetSelection

The text selection.

kTextBackColor

The text background color.

kTextForeColor

The text foreground color.

TextMediaHiliteTextSample

kTextFace
The text face.

kTextFont
The text font.

kTextSize
The text size.

kTextAlignment
The text alignment.

kTextHilite
The text highlighting.

kTextDropShadow
The text drop shadow.

kTextDisplayFlags
Display flags.

kTextScroll
The text scroll position.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: Movies.h

TextMediaHiliteTextSample

Specifies selected text to be highlighted for a given text media handler.

```
ComponentResult TextMediaHiliteTextSample (
    MediaHandler      mh,
    TimeValue         sampleTime,
    short             hiliteStart,
    short             hiliteEnd,
    RGBColor          *rgbHiliteColor );
```

mh
The text media handler obtained by GetMediaHandler (I-432).

sampleTime
The starting time in the sample.

`hiliteStart`

The beginning of the text to be highlighted.

`hiliteEnd`

The end of the text to be highlighted. If the `hiliteEnd` parameter is greater than the `hiliteStart` parameter, then the text is highlighted from the selection specified by `hiliteStart` to `hiliteEnd`.

`rgbHiliteColor`

Contains a pointer to an `RGBColor` (IV–2403) structure that defines the color for highlighting. If this parameter is not `NIL`, then the specified color is used when highlighting the text indicated by the `hiliteStart` and `hiliteEnd` parameters. Otherwise, the default system highlight color is used.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Text Media Handler Functions” (V–2756)

Related Java Methods

`quicktime.std.movies.media.TextMediaHandler.hiliteTextSample()`

TextMediaRawIdle

Undocumented

```
ComponentResult TextMediaRawIdle (
    MediaHandler    mh,
    GWorldPtr       gw,
    GDHandle        gd,
    TimeValue       sampleTime,
    long            flagsIn,
    long            *flagsOut );
```

`mh`

The text media handler obtained by `GetMediaHandler` (I–432).

`gw`

A pointer to a `CGrafPort` (IV–2168) structure that defines a graphics world.

TextMediaRawSetup

gd

A handle to a graphics device.

sampleTime

Undocumented

flagsIn

Undocumented

flagsOut

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

flagsIn Constants

Undocumented

Undocumented

flagsOut Constants

Undocumented

Undocumented

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

TextMediaRawSetup

Undocumented

```
ComponentResult TextMediaRawSetup (
    MediaHandler      mh,
    GWorldPtr         gw,
    GDHandle          gd,
    void              *data,
    long              dataSize,
    TextDescriptionHandle tdh,
    TimeValue         sampleDuration );
```

mh

The text media handler obtained by GetMediaHandler (I-432).

gw

A pointer to a CGrafPort (IV–2168) structure that defines a graphics world.

gd

A handle to a graphics device.

data

A pointer to data.

dataSize

The size of the data.

tdh

A handle to a TextDescription (IV–2474) structure.

sampleDuration

Undocumented

function result You can access Movie Toolbox error returns through GetMoviesError (I–492) and GetMoviesStickyError (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

TextMediaSetTextProc

Specifies a custom function to be called whenever a text sample is displayed in a movie.

```
ComponentResult TextMediaSetTextProc (
    MediaHandler      mh,
    TextMediaUPP      TextProc,
    long               refcon );
```

mh

The text media handler obtained by GetMediaHandler (I–432).

TextProc

A **Universal Procedure Pointer** that points to a TextMediaProc (III–2150) callback.

TextMediaSetTextProperty

refcon

Indicates a reference constant that will be passed to your callback. Use this parameter to point to a data structure containing any information your function needs. Set this parameter to 0 if you don't need it.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Text Media Handler Functions” (V-2756)

TextMediaSetTextProperty

Sets properties of a text media.

```
ComponentResult TextMediaSetTextProperty (
    MediaHandler    mh,
    TimeValue       atMediaTime,
    long            propertyType,
    void            *data,
    long            dataSize );
```

mh

The text media handler obtained by `GetMediaHandler` (I-432).

atMediaTime

The media time of the text.

propertyType

A constant (see below) that identifies the text property to be set.

data

A pointer to text data.

dataSize

The size of the data.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494), as well as in the function result. See “Error Codes” (IV-2663).

propertyType Constants

kTextTextHandle	A handle to the text.
kTextTextPtr	A pointer to the text.
kTextTStyle	The text style.
kTextSetSelection	The text selection.
kTextBackColor	The text background color.
kTextForeColor	The text foreground color.
kTextFace	The text face.
kTextFont	The text font.
kTextSize	The text size.
kTextAlignment	The text alignment.
kTextHilite	The text highlighting.
kTextDropShadow	The text drop shadow.
kTextDisplayFlags	Display flags.
kTextScroll	The text scroll position.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

TextMediaSetTextSampleData

TextMediaSetTextSampleData

Sets values before calling `TextMediaAddTextSample` (III–1936) or `TextMediaAddTESample` (III–1933).

```
ComponentResult TextMediaSetTextSampleData (  
    MediaHandler    mh,  
    void            *data,  
    OSType          dataType );
```

`mh`

A reference to the text media handler. You obtain this reference from `GetMediaHandler` (I–432).

`data`

A pointer to the data, defined by the `dataType` parameter.

`dataType`

The type of data.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

The following code sample demonstrates how to use this function:

```
// TextMediaSetTextSampleData coding example  
short          trans = 127;  
Point          dropOffset;  
MediaHandler    mh;  
  
dropOffset.h = dropOffset.v = 4  
TextMediaSetTextSampleData(mh, (void *)&dropOffset, dropShadowOffsetType);  
TextMediaSetTextSampleData(mh, (void *)&trans, dropShadowTranslucencyType);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Text Media Handler Functions” (V–2756)

Related Java Methods

```
quicktime.std.movies.media.TextMediaHandler.setDropShadowOffset(),  
quicktime.std.movies.media.TextMediaHandler.setDropShadowTranslucency()
```

TrackTimeToMediaTime

Converts a track's time value to a time value that is appropriate to the track's media, using the track's edit list.

```
TimeValue TrackTimeToMediaTime (
    TimeValue    value,
    Track        theTrack );
```

value

The track's time value; must be expressed in the time scale of the movie that contains the track.

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

function result The track's time value, but in the media's time coordinate system. If the track time corresponds to empty space, this function returns a value of –1.

Discussion

This function maps the track time through the track's edit list to come up with the media time. This time value contains the track's time value according to the media's time coordinate system. If the time you specified lies outside of the movie's active segment or corresponds to empty space in the track, this function returns a value of –1. Hence you can use it to determine whether a specified track edit is empty.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Working With Track Time” (V–2733)

Related Java Methods

`quicktime.std.movies.Track.trackTimeToMediaTime()`

TransformFixedPoints

Transforms a set of fixed points through a specified matrix.

```
OSErr TransformFixedPoints (
    const MatrixRecord    *m,
    FixedPoint            *fpt,
```

TransformFixedRect

```
long count );
```

m

A pointer to the transformation matrix for this operation.

fpt

A pointer to the first fixed point to be transformed.

count

The number of fixed points to be transformed. These points must be stored immediately following the point specified by the *fpt* parameter.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: *ImageCompression.h*

Programming summary: “Applying Matrix Transformations” (V–2766)

Related Java Methods

quicktime.std.image.Matrix.transformDPoints()

TransformFixedRect

Transforms the upper-left and lower-right points of a rectangle through a matrix that is specified by fixed points.

```
Boolean TransformFixedRect (  
    const MatrixRecord    *m,  
    FixedRect             *fr,  
    FixedPoint             *fpp );
```

m

A pointer to the matrix for this operation.

fr

A pointer to the *FixedRect* (IV–2260) structure that defines the rectangle to be transformed. *TransformFixedRect* returns the updated coordinates into the structure referred to by this parameter. If the resulting rectangle has been rotated or skewed (that is, the transformation involves operations other than scaling and translation), the function sets the returned *Boolean* value to *FALSE* and returns the coordinates of the boundary box of the transformed rectangle.

The function then updates the points specified by the `fpp` parameter to contain the coordinates of the four corners of the transformed rectangle.

`fpp`

A pointer to an array of four fixed points. The function returns the coordinates of the four corners of the rectangle after the transformation operation. If you do not want this information, set this parameter to `NIL`.

function result If the resulting rectangle has been rotated or skewed (that is, the transformation involves operations other than scaling and translation), the function returns `FALSE`, updates the rectangle specified by the `fr` parameter to define the boundary box of the resulting rectangle, and places the coordinates of the corners of the resulting rectangle in the points specified by the `fpp` parameter. If the transformed rectangle and its boundary box are the same, the function returns `TRUE`.

Discussion

This function does not return any error codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Applying Matrix Transformations” (V-2766)

Related Java Methods

`quicktime.std.image.Matrix.transformDRect()`

TransformPoints

Transforms a set of QuickDraw points through a specified matrix.

```
OSErr TransformPoints (
    const MatrixRecord *mp,
    Point *pt1,
    long count );
```

`mp`

A pointer to the transformation matrix for this operation.

`pt1`

A pointer to the first QuickDraw point to be transformed.

TransformRect

count

The number of QuickDraw points to be transformed. These points must be stored immediately following the point specified by the *pt1* parameter.

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Applying Matrix Transformations” (V–2766)

Related Java Methods

`quicktime.std.image.Matrix.transformPoints()`

TransformRect

Transforms the upper-left and lower-right points of a rectangle through a specified matrix.

```
Boolean TransformRect (  
    const MatrixRecord    *m,  
    Rect                  *r,  
    FixedPoint            *fpp );
```

m

The matrix for this operation.

r

A pointer to the *Rect* (IV–2399) structure that defines the rectangle to be transformed. The function returns the updated coordinates into the structure referred to by this parameter.

fpp

A pointer to an array of four fixed points. The *TransformRect* function returns the coordinates of the four corners of the rectangle after the transformation operation. If you do not want this information, set this parameter to *NIL*.

function result If the resulting rectangle has been rotated or skewed (that is, the transformation involves operations other than scaling and translation), the function returns *FALSE*, updates the rectangle specified by the *r* parameter to define the boundary box of the resulting rectangle, and places the coordinates of the corners of the resulting rectangle in the points specified by the *fpp* parameter. If the

transformed rectangle and its boundary box are the same, the function returns TRUE.

Discussion

This function does not return any error codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Applying Matrix Transformations” (V–2766)

Related Java Methods

`quicktime.std.image.Matrix.transformRect()`

TransformRgn

Applies a specified matrix to a region.

```
OSErr TransformRgn (
    MatrixRecordPtr    matrix,
    RgnHandle          rgn );
```

matrix

Points to the matrix for this operation. The `TransformRgn` function currently supports only translation and scaling operations.

rgn

A handle to the `MacRegion` (IV–2303) structure to be transformed. The function transforms each point in the region according to the specified matrix

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Applying Matrix Transformations” (V–2766)

Related Java Methods

`quicktime.std.image.Matrix.transformRgn()`

TranslateMatrix

TranslateMatrix

Adds a translation value to a specified matrix.

```
void TranslateMatrix (
    MatrixRecord    *m,
    Fixed           deltaH,
    Fixed           deltaV );
```

m

A pointer to the MatrixRecord (IV-2304) structure for this operation.

deltaH

The value to be added to the x coordinate translation value.

deltaV

The value to be added to the y coordinate translation value.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

Programming summary: “Managing Matrices” (V-2764)

Related Java Methods

quicktime.std.image.Matrix.translate()

TrimImage

Adjusts a compressed image to the boundaries defined by a specified rectangle.

```
OSErr TrimImage (
    ImageDescriptionHandle    desc,
    Ptr                      inData,
    long                     inBufferSize,
    ICMDataProcRecordPtr     dataProc,
    Ptr                      outData,
    long                     outBufferSize,
    ICMFlushProcRecordPtr    flushProc,
    Rect                     *trimRect,
    ICMProgressProcRecordPtr progressProc );
```

desc

A handle to the `ImageDescription` (IV–2285) structure that describes the compressed image. On return, the compressor updates this structure to describe the resized image.

inData

A pointer to the compressed image data. This pointer must contain a **32-bit clean** address. If the entire compressed image cannot be stored at this location, your application may provide an `ICMDataProc` callback through the `dataProc` parameter.

inBufferSize

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If you have not specified a data-loading function, this parameter is ignored.

dataProc

A pointer to an `ICMDataProcRecord` (IV–2279) structure that references an `ICMDataProc` (III–2090) callback. If there is not enough memory to store the compressed image, the compressor calls a function you provide that loads more compressed data. If you have not provided such a data-loading function, set this parameter to `NIL`. In this case, the compressor expects that the entire compressed image is in the memory location specified by the `inData` parameter.

outData

A pointer to a buffer to receive the trimmed image. The Image Compression Manager places the actual size of the resulting image into the `dataSize` field of the `ImageDescription` (IV–2285) structure referred to by the `desc` parameter. This pointer must contain a **32-bit clean** address. Your application should create a destination buffer at least as large as the source image. If there is not sufficient memory to store the compressed image, you may choose to write the compressed data to mass storage during the compression operation, in which case you use the `flushProc` parameter to identify your data-unloading function to the compressor.

outBufferSize

The size of the buffer to be used by the data-unloading function specified by the `flushProc` parameter. If you have not specified a data-unloading function, this parameter is ignored.

flushProc

A pointer to an `ICMFlushProcRecord` (IV–2280) structure that references an `ICMFlushProc` (III–2091) callback. If there is not enough memory to store the compressed image, the compressor calls a function you provide that unloads some of the compressed data. If you have not provided such a data-unloading

TuneGetIndexedNoteChannel

function, set this parameter to `NIL`. In this case, the compressor writes the entire compressed image into the memory location specified by the `data` parameter

`trimRect`

A pointer to a `Rect` (IV–2399) structure that defines the desired image dimensions. On return, the function adjusts the rectangle values so that they refer to the same rectangle in the result image. This is necessary whenever data is removed from the beginning or left side of the image.

`progressProc`

A pointer to an `ICMProgressProcRecord` (IV–2284) structure that references an `ICMProgressProc` (III–2093) callback. During the operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided such a **progress function**, set this parameter to `NIL`. If you pass a value of `-1`, you obtain a standard progress function.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `ImageCompression.h`

Programming summary: “Working With Pixel Maps” (V–2789)

Related Java Methods

`quicktime.std.image.QTImage.trim()`

TuneGetIndexedNoteChannel

Determines how many parts a tune is playing and which instrument is assigned to those parts.

```
ComponentResult TuneGetIndexedNoteChannel (
    TunePlayer      tp,
    long            i,
    NoteChannel     *nc );
```

`tp`

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`i`

Note channel index, or 0 to get the number of parts.

nc

A pointer to an allocated initialized note channel.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

The tune player allocates note channels that best satisfy the requested instrument in the tune header. The application can use this call to determine which instrument was actually used for each note channel. This function takes the tune player in the tp parameter and returns the number of parts (1...n) allocated to the tune player. You can then pass the function a part index and it returns, in the nc parameter, the note channel allocated for that part.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

quicktime.std.music.TunePlayer.getNumberOfNoteChannels(),
quicktime.std.music.TunePlayer.getIndexedNoteChannel()

TuneGetNoteAllocator

Returns the instance of the note allocator that the tune player is using.

```
NoteAllocator TuneGetNoteAllocator (
    TunePlayer    tp );
```

tp

A tune player identifier, obtained from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

function result A note allocator or an error code. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeMusic.h

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

quicktime.std.music.TunePlayer.getNoteAllocator()

TuneGetPartMix

TuneGetPartMix

Gets volume, balance, and mixing settings for a specified part of a tune.

```
ComponentResult TuneGetPartMix (
    TunePlayer      tp,
    unsigned long    partNumber,
    long            *volumeOut,
    long            *balanceOut,
    long            *mixFlagsOut );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

partNumber

The part number for this request.

volumeOut

Returns the volume for the part.

balanceOut

Returns the balance for the part.

mixFlagsOut

Returns flags (see below) that control part mixing.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

mixFlagsOut Constants

`kTuneMixMute`

Disable the part so that it is not heard.

`kTuneMixSolo`

Include only soloed parts in the mix if any parts are soloed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.getPartMix()`

See Also

For the corresponding set function, see `TuneSetPartMix` (III–1970).

TuneGetStatus

Returns an initialized structure describing the state of the tune player instance.

```
ComponentResult TuneGetStatus (
    TunePlayer    tp,
    TuneStatus    *status );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

status

A pointer to an initialized `TuneStatus` (IV–2492) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.getStatus()`

TuneGetTimeBase

Returns the time base of the tune player.

```
ComponentResult TuneGetTimeBase (
    TunePlayer    tp,
    TimeBase      *tb );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

tb

A pointer to a time base identifier, such as that returned by `NewTimeBase` (II–1119). On return, the time base used to control the sequence timing.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

TuneGetTimeScale

Discussion

The sequence can be controlled in several ways through its time base. The rate of playback can be changed, or the time base object can be slaved to a clock or time base different than real time.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

```
quicktime.std.music.TunePlayer.getTimeBase(),  
quicktime.std.clocks.TimeBase.fromTunePlayer(),  
quicktime.std.clocks.TimeBase.fromSequenceGrabber()
```

TuneGetTimeScale

Returns the current time scale for a specified tune player instance.

```
ComponentResult TuneGetTimeScale (  
    TunePlayer    tp,  
    TimeScale     *scale );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

scale

A pointer to an initialized `TimeScale` variable that indicates the tune player’s current time scale in units per second.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

```
quicktime.std.music.TunePlayer.getTimeScale()
```

See Also

For the corresponding set function, see `TuneSetTimeScale` (III–1973).

TuneGetVolume

Returns the volume associated with an entire tune sequence.

```
ComponentResult TuneGetVolume (
    TunePlayer    tp );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result The volume as a value from 0.0 to 1.0, or a negative result code. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.getVolume()`

See Also

For the corresponding set function, see `TuneSetVolume` (III–1974).

TuneInstant

Plays a particular sequence of events active at a specified position.

```
ComponentResult TuneInstant (
    TunePlayer    tp,
    unsigned long *tune,
    unsigned long  tunePosition );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

tune

A pointer to tune sequence data.

tunePosition

The position within the tune sequence data in time units.

TunePreroll

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function plays the notes that are “on” at the point specified by the `tunePosition` parameter. The notes are started and then left playing on return. The notes can be silenced by calling `TuneStop` (III–1975). This call is useful for enabling user “scrubbing” on a sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.instant()`

TunePreroll

Prepares to play a tune player sequence data by attempting to reserve note channels for each part in the sequence.

```
ComponentResult TunePreroll (  
    TunePlayer    tp );
```

`tp`

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.preroll()`

TuneQueue

Places a sequence of music events into a queue to be played.

```

ComponentResult TuneQueue (
    TunePlayer          tp,
    unsigned long       *tune,
    Fixed               tuneRate,
    unsigned long       tuneStartPosition,
    unsigned long       tuneStopPosition,
    unsigned long       queueFlags,
    TuneCallbackUPP     callBackProc,
    long                refCon );

```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

tune

A pointer to an array of events, terminated by a marker event of subtype `kMarkerEventEnd`. See “QTMA Events” (IV–2686).

tuneRate

Speed at which to play the sequence. “Normal” speed is `0x00010000`.

tuneStartPosition

Sequence starting time.

tuneStopPosition

Sequence stopping time. The `tuneStartPosition` and `tuneStopPosition` parameters specify, in time units numbered from 0 for the beginning of the sequence, which part of the queued sequence to play. To play all of it, pass 0 and `0xFFFFFFFF`, respectively.

queueFlags

Flags (see below) with details about how to play the queued tunes.

callBackProc

A pointer to a `TuneCallbackProc` (III–2152) callback.

refCon

A reference constant to be passed to your `TuneCallbackProc` callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

queueFlags Constants

kTuneStartNow

Play even if another tune is playing. If there is a sequence currently playing, the newly queued sequence begins as soon as the active sequence ends unless the

TuneSetBalance

`kTuneStartNow` flag is set, in which case the currently playing sequence is immediately terminated and the new one started.

`kTuneDontClipNotes`

Allow notes to finish their durations outside sample.

`kTuneExcludeEdgeNotes`

Don't play notes that start at the end of the tune.

`kTuneQuickStart`

Leave all the controllers where they are and ignore the start time.

`kTuneLoopUntil`

Loop a queued tune if there is nothing else in the queue.

`kTuneStartNewMaster`

Start a new master reference timer.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.queue()`

TuneSetBalance

Modifies the pan controller setting for a tune player.

```
ComponentResult TuneSetBalance (
    TunePlayer    tp,
    long          balance );
```

`tp`

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`balance`

A new pan controller setting. Valid values are from –128 to 128 for left-to-right balance.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming InfoC interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods`quicktime.std.music.TunePlayer.setBalance()`**TuneSetHeader**

Prepares the tune player to accept subsequent music event sequences by defining one or more parts to be used by sequence Note events.

```
ComponentResult TuneSetHeader (
    TunePlayer      tp,
    unsigned long    *header );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

header

A pointer to a list of instruments that will be used in subsequent calls to the `TuneQueue` function. The list can include events with subtypes of `kGeneralEventNoteRequest`, `kGeneralEventPartKey`, `kGeneralEventAtomicInstrument`, `kGeneralEventMIDIChannel`, and `kGeneralEventUsedNotes`. It can also include atomic instruments. The list is terminated by a marker event of subtype `kMarkerEventEnd`. See “QTMA Events” (IV–2686).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is the first QuickTime music architecture call to play a music sequence. The header parameter points to one or more initialized General events and atomic instruments. Only one call to this function is required. Each call to this function resets the tune player.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming InfoC interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

TuneSetHeaderWithSize

Related Java Methods

`quicktime.std.music.TunePlayer.setHeader()`

See Also

See `TuneSetHeaderWithSize` (III–1968) and `TuneSetNoteChannels` (III–1969).

TuneSetHeaderWithSize

Similar to `TuneSetHeader` (III–1967) but lets you specify the header length.

```
ComponentResult TuneSetHeaderWithSize (
    TunePlayer      tp,
    unsigned long   *header,
    unsigned long    size );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

header

A pointer to a list of instruments that will be used in subsequent calls to the `TuneQueue` function. The list can include events with subtypes of `kGeneralEventNoteRequest`, `kGeneralEventPartKey`, `kGeneralEventAtomicInstrument`, `kGeneralEventMIDIChannel`, and `kGeneralEventUsedNotes`. It can also include atomic instruments. The list is terminated by a marker event of subtype `kMarkerEventEnd`. See “QTMA Events” (IV–2686).

size

The size of the header in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function resembles `TuneSetHeader` (III–1967) in that it prepares the tune player to accept subsequent music event sequences by defining one or more parts to be used by sequence Note events. But unlike `TuneSetHeader`, it allows you to specify the header length in bytes. This prevents the call from parsing off the end if the music event sequence is missing an end marker.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.setHeaderWithSize()`

See Also

See `TuneSetHeader` (III–1967) and `TuneSetNoteChannels` (III–1969).

TuneSetNoteChannels

Assigns note channels to a tune player.

```
ComponentResult TuneSetNoteChannels (
    TunePlayer          tp,
    unsigned long       count,
    NoteChannel         *noteChannelList,
    TunePlayCallbackUPP playCallbackProc,
    long                refCon );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

count

The number of note channels to assign.

noteChannelList

A pointer to the list of note channels to assign. The parts for the note channels you assign are numbered from 1 to the value of the *count* parameter.

playCallbackProc

A pointer to a `TunePlayCallbackProc` (III–2152) callback that is called for each event whose part number is greater than the value of the *count* parameter. Events whose part numbers are less than or equal to the value of the *count* parameter are passed to the note channel rather than the callback. This lets you to use the tune player as a general purpose timer/sequencer.

refCon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

TuneSetPartMix

Discussion

When you call this function, any note channels that were previously assigned to the tune player are no longer used and are disposed of.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.setNoteChannels()`

TuneSetPartMix

Sets volume, balance, and mixing settings for a specified part of a tune.

```
ComponentResult TuneSetPartMix (
    TunePlayer      tp,
    unsigned long    partNumber,
    long            volume,
    long            balance,
    long            mixFlags );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

partNumber

The part number for this request.

volume

The volume for the part.

balance

The balance for the part.

mixFlags

Flags (see below) that control part mixing.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

mixFlags Constants

`kTuneMixMute`

Disable the part so that it is not heard.

kTuneMixSolo

Include only soloed parts in the mix if any parts are soloed.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.setPartMix()`

See Also

For the corresponding get function, see `TuneGetPartMix` (III–1960).

TuneSetPartTranspose

Modifies the pitch and volume of every note of a tune.

```
ComponentResult TuneSetPartTranspose (
    TunePlayer      tp,
    unsigned long    part,
    long             transpose,
    long             velocityShift );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

part

The part for which you want to change pitch and volume.

transpose

A value by which to modify the pitch of the note. The value is a small integer for semitones or an 8.8 fixed-point number for microtones.

velocityShift

A value to add to the `velocity` parameter passed to `NAPlayNote` (II–1036).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

TuneSetSofter

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.setPartTranspose()`

TuneSetSofter

Adjusts the volume a tune is played at to the softer volume produced by QuickTime 2.1.

```
ComponentResult TuneSetSofter (
    TunePlayer    tp,
    long          softer );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

softer

A value of 1 means play at the QuickTime 2.1 volume; a value of 0 means don’t make the volume softer.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function adjusts the volume a tune is played at to the softer volume produced by QuickTime 2.1. Files imported with QuickTime 2.1 automatically play softer. Files imported with QuickTime 2.5 or later play at the new, louder volume.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.setSofter()`

TuneSetSoundLocalization

Passes sound localization data to a tune player.

```
ComponentResult TuneSetSoundLocalization (
    TunePlayer    tp,
    Handle        data );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

data

The sound localization data to be passed.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.setSoundLocalization()`

TuneSetTimeScale

Sets the time scale used by the specified tune player instance.

```
ComponentResult TuneSetTimeScale (
    TunePlayer    tp,
    TimeScale     scale );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

scale

The time scale value to be used, in units per second.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function sets the time scale data used by the tune player’s sequence data when interpreting time-based events.

Version Notes

Introduced in QuickTime 3 or earlier.

TuneSetVolume

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.setTimeScale()`

See Also

For the corresponding get function, see `TuneGetTimeScale` (III–1962).

TuneSetVolume

Sets the volume for an entire sequence.

```
ComponentResult TuneSetVolume (  
    TunePlayer    tp,  
    Fixed         volume );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

volume

The volume to use for the sequence. The value is a fixed 16.16 number.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function sets the volume level of the active sequence to the value of the `volume` parameter, ranging from 0.0 to 1.0. Individual instruments within the sequence can maintain independent volume levels.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.setVolume()`

See Also

For the corresponding get function, see `TuneGetVolume` (III–1963).

TuneStop

Stops a currently playing sequence.

```
ComponentResult TuneStop (
    TunePlayer    tp,
    long          stopFlags );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

stopFlags

Set to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.stop()`

TuneTask

Lets a tune player to perform tasks it must perform at foreground task time.

```
ComponentResult TuneTask (
    TunePlayer    tp );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Call this function periodically to allow a tune player to perform certain operations it can performed only at foreground application task time. Specifically, the QuickTime music synthesizer cannot load instruments from disk at interrupt time.

TuneUnroll

As a result, embedded program changes are not performed until this function is called.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.task()`

TuneUnroll

Releases any note channel **resources** that may have been locked down by previous calls to `TunePreroll` (III–1964) for this tune player.

```
ComponentResult TuneUnroll (  
    TunePlayer    tp );
```

tp

A tune player identifier, obtained from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeMusic.h`

Programming summary: “Using the Tune Player” (V–2914)

Related Java Methods

`quicktime.std.music.TunePlayer.unroll()`

TweenerDoTween

Performs a **tween** operation.

```
ComponentResult TweenerDoTween (  
    TweenerComponent    tc,  
    TweenRecord          *tr );
```

tc

The **tween** component for this operation.

tr

A pointer to the TweenRecord (IV–2493) structure for the **tween** operation.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

QuickTime calls this function to interpolate the data used during a **tween** operation. The TweenRecord (IV–2493) structure contains complete information about the tween operation, including the start and end values for the operation and a percentage that indicates the progress towards completion of the tween sample. This function should use the information in the tween record to calculate the tweened value, and should call the data function specified in the tween record, passing it the tweened value.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Tween Component Requirements” (V–2876)

TweenerInitialize

Initializes your **tween** component for a single tween operation.

```
ComponentResult TweenerInitialize (
    TweenerComponent    tc,
    QTAtomContainer     container,
    QTAtom              tweenAtom,
    QTAtom              dataAtom );
```

tc

The **tween** component for this operation.

container

The container that holds the atoms specified by the **tweenAtom** and **dataAtom** parameters.

tweenAtom

The atom that contains all parameters for defining this **tween**. This includes the data atom and any special atoms, such as an atom of type 'qdrq' (IV–2574), that may be necessary.

TweenerReset

`dataAtom`

The atom that contains the values to be **tweened**. This atom is a child of the atom specified by the `tweenAtom` parameter. This parameter is provided as a convenience; you can also call QT atom container functions to locate the data atom in the container.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function sets up the **tween** component when it is first used. In your implementation of this function, you can allocate storage and set up any structures that you need for the duration of a tween operation. Although the container that holds the data atom is available during each call to `TweenerDoTween` (III–1976), you can improve the performance of your tween component by extracting the data to be used by the `TweenerDoTween` function in this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Tween Component Requirements” (V–2876)

TweenerReset

Cleans up when the **tween** operation is finished.

```
ComponentResult TweenerReset (  
    TweenerComponent    tc );
```

`tc`

The **tween** component for this operation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function releases storage allocated by the **tween** component when the component is no longer being used. It should release any storage allocated by the `TweenerInitialize` (III–1977) function and close or release any other **resources** used by the component. A tween component may receive a `TweenerInitialize` call after being reset.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Tween Component Requirements” (V–2876)

U

UncaptureComponent

Uncaptures a previously captured component.

```
OSErr UncaptureComponent (
    Component    aComponent );
```

`aComponent`

The component identifier of the component to be uncaptured. Your component obtains this identifier from `CaptureComponent` (I–74).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Components.h`

Programming summary: “Functions Used By Components” (V–2782)

UnregisterComponent

Removes a component from the Component Manager’s registration list.

```
OSErr UnregisterComponent (
    Component    aComponent );
```

`aComponent`

A component identifier that specifies the component to be removed. Applications that register components may obtain this identifier from `RegisterComponent` (III–1451) or `RegisterComponentResource` (III–1453). The component to be removed from the registration list must not be in use by any applications or components.

UnsignedFixedMulDiv

function result See “Error Codes” (IV–2663). If there are open connections to the component, this function returns a negative result code. Returns noErr if there is no error.

Discussion

Most components are registered at startup and remain registered until the computer is shut down. However, you may want to provide some services temporarily. In that case you dispose of the component that provides the temporary service by using this function. It removes the component with the specified component identifier from the list of available components.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Components.h

Programming summary: “Functions Used By Components” (V–2782)

UnsignedFixedMulDiv

Performs multiplications and divisions on unsigned fixed-point numbers.

```
UnsignedFixed UnsignedFixedMulDiv (  
    UnsignedFixed    value,  
    UnsignedFixed    multiplier,  
    UnsignedFixed    divisor );
```

value

The value to be multiplied or divided.

multiplier

The multiplier to be applied to the value in the *value* parameter. Pass 0x00010000 if you do not want to multiply.

divisor

The divisor to be applied to the value in the *value* parameter. Pass 0x00010000 if you do not want to divide.

function result The fixed-point number that is the value of the *value* parameter, multiplied by the value in the *multiplier* parameter and divided by the value in the *divisor* parameter. The function performs both operations before returning.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: Sound.h

UnsignedFixMulDiv

Performs multiplications and divisions on unsigned fixed-point numbers.

```
Fixed UnsignedFixMulDiv (  
    Fixed    src,  
    Fixed    mul,  
    Fixed    divisor );
```

src

The value to be multiplied or divided.

mul

The multiplier to be applied to the value in the *src* parameter. Pass 0x00010000 if you do not want to multiply.

divisor

The divisor to be applied to the value in the *src* parameter. Pass 0x00010000 if you do not want to divide.

function result The fixed-point number that is the value of the *src* parameter, multiplied by the value in the *mul* parameter and divided by the value in the *divisor* parameter. The function performs both operations before returning.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: ImageCompression.h

UpdateMovie

Ensures that the Movie Toolbox properly displays your movie after it has been uncovered.

```
OSErr UpdateMovie (  
    Movie    theMovie );
```

UpdateMovie

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

Your application should call this function during window updating. Don’t call `MoviesTask` (II–973) at this time; you will observe better display behavior if you call it at the end of your update processing.

This function does not actually update the movie’s graphics world. Rather, it invalidates the movie’s display state so that the Movie Toolbox redraws the movie the next time you call `MoviesTask`. If you need to force a movie to be redrawn outside of a window update sequence, your application can call this function and then call `MoviesTask` to service the movie. The Movie Toolbox determines the portion of the screen to update by examining the graphics port’s visible region.

The following code snippet uses this function in a Macintosh Window Manager update sequence:

```
// UpdateMovie coding example
#include <Events.h>
#include <ToolUtils.h>
#include "Movies.h"

void DoUpdate (WindowPtr theWindow, Movie theMovie)
{
    BeginUpdate (theWindow);
    UpdateMovie (theMovie);
    EndUpdate (theWindow);
} /* DoUpdate */
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Movies and Your Event Loop” (V–2720)

Related Java Methods

```
quicktime.std.movies.Movie.update()
```

UpdateMovieResource

Replaces the contents of a movie **resource** in a specified movie file.

```
OSErr UpdateMovieResource (
    Movie          theMovie,
    short          resRefNum,
    short          resId,
    ConstStr255Param resName );
```

theMovie

The movie you wish to place in the movie file. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

resRefNum

Identifies the movie file that contains the **resource** to be changed. Your application obtains this value from `OpenMovieFile` (II–1133).

resId

The **resource** to be changed. This value is obtained from a previous call to `NewMovieFromFile` (II–1080), `NewMovieFromDataRef` (II–1078), or (eng>or–1078) If you specify a single-fork movie file by passing the `movieInDataForkResID` constant, the Movie Toolbox places the movie resource into the file's data fork.

resName

Points to a new name for the **resource**. If you don't want to change the resource's name, set this parameter to `NIL`.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Discussion

You specify the movie that is to be placed into the **resource**. This function can accommodate single-fork movie files. After updating the movie file, this function clears the movie changed flag.

Version Notes

Introduced in QuickTime 3 or earlier.

UseMovieEditState

Programming Info

C interface file: `Movies.h`

Programming summary: “Saving Movies” (V–2715)

Related Java Methods

`quicktime.std.movies.Movie.updateResource()`

UseMovieEditState

Returns a movie to the condition determined by an edit state created previously.

```
OSErr UseMovieEditState (
    Movie          theMovie,
    MovieEditState toState );
```

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

toState

The edit state for this operation. Your application obtains this edit state identifier when you create the edit state by calling `NewMovieEditState` (II–1073).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Undo for Movies” (V–2748)

Related Java Methods

`quicktime.std.movies.Movie.useEditState()`

UseTrackEditState

Returns a track to the condition determined by an edit state created previously.

```
OSErr UseTrackEditState (
    Track          theTrack,
```

```
TrackEditState state );
```

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

state

The edit state for this operation. Your application obtains this edit state identifier when you create the edit state by calling `NewTrackEditState` (II–1119).

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Undo for Tracks” (V–2751)

Related Java Methods

```
quicktime.std.movies.Track.useEditState()
```

V

V

V DAddKeyColor

Adds a key color to a component’s list of active key colors.

```
VideoDigitizerError VAddKeyColor (
    VideoDigitizerComponent ci,
    long *index );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

index

A pointer to the color to add to the key color list. The value of the `index` field corresponds to a color in the current **color lookup table**.

VDClearClipRgn

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Selectively Displaying Video” (V–2852)

VDClearClipRgn

Disables all or part of a clipping region that was previously set with VDSaveClipRgn (III–2032).

```
VideoDigitizerError VDClearClipRgn (  
    VideoDigitizerComponent    ci,  
    RgnHandle                   clipRegion );
```

ci

The video digitizer component for the request. Applications obtain this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

clipRegion

A handle to a MacRegion (IV–2303) structure that defines the clipping region to clear. This region must correspond to all or part of the clipping region established previously with VDSaveClipRgn (III–2032).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Video Clipping” (V–2853)

Related Java Methods

quicktime.std.sg.VideoDigitizer.clearClipRgn()

VDCompressDone

Determines whether the video digitizer has finished digitizing and compressing a frame of image data.


```

VideoDigitizerError VDCompressDone (
    VideoDigitizerComponent    ci,
    Boolean                    *done,
    Ptr                        *theData,
    long                      *dataSize,
    UInt8                     *similarity,
    TimeRecord                 *t );

```

ci

Identifies the application’s connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

done

A pointer to a `Boolean` value. Video digitizers set this value to `TRUE` when they are done. If the digitizer is not yet done, it sets the `Boolean` value to `FALSE`. In this case, the digitizer does not return any other information. The digitizer is careful to return the frames in temporal order, and to avoid returning two frames with the same time value (unless the rate is set to 0).

theData

A pointer to a field that is to receive a pointer to the compressed image data. The digitizer returns a pointer that is valid in the application’s current memory mode.

dataSize

A pointer to a field to receive a value indicating the number of bytes of compressed image data.

similarity

A pointer to a field to receive an indication of the relative similarity of this image to the previous image in a sequence. A value of 0 indicates that the current frame is a **key frame** in the sequence. A value of 255 indicates that the current frame is identical to the previous frame. Values from 1 through 254 indicate relative similarity, ranging from very different (1) to very similar (254). This field is only filled in if the temporal quality passed in with `VDSetCompression` (III–2034) is not 0; that is, if it is not frame-differenced.

t

A pointer to a `TimeRecord` (IV–2486) structure. When the operation is complete, the digitizer fills in this structure with information indicating when the frame was grabbed. The time value stored in this structure is in the time base that the application sets with `VDSetTimeBase` (III–2054).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

VDCompressOneFrameAsync

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Compressed Source Devices” (V–2846)

VDCompressOneFrameAsync

Instructs the video digitizer to digitize and compress a single frame of image data.

```
VideoDigitizerError VDCompressOneFrameAsync (  
    VideoDigitizerComponent    ci );
```

ci

Identifies the application’s connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Unlike `VDGrabOneFrameAsync` (III–2025), this function causes the video digitizer to handle all details of managing data buffers.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Compressed Source Devices” (V–2846)

VDDone

Determines if `VDGrabOneFrameAsync` (III–2025) is finished with a specific output buffer.

```
VideoDigitizerError VDDone (  
    VideoDigitizerComponent    ci,  
    short                      buffer );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

buffer

Identifies the buffer for the operation. The value of this parameter must correspond to a valid index into the list of buffers you supply when your application calls `VDSetupBuffers` (III–2055). This value is zero-based; that is, you must set this parameter to 0 to refer to the first buffer in the buffer list.

function result

Returns a long integer indicating whether the specified asynchronous frame grab is complete. If the returned value is 0, the video digitizer component is still working on the frame. If the returned value is nonzero, the digitizer component is finished with the frame and the application can perform its processing.

Discussion

Applications can determine whether a video digitizer component supports asynchronous frame grabbing by examining the output capability flags of the digitizer component, using `VDGetCurrentFlags` (III–1996). Specifically, if the `digiOutDoesAsyncGrabs` flag is set to 1, the digitizer component supports both this function and `VDGrabOneFrameAsync` (III–2025).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Digitization” (V–2848)

VDGetActiveSrcRect

Obtains size and location information for the active source rectangle used by a video digitizer component.

```
VideoDigitizerError VDGetActiveSrcRect (
    VideoDigitizerComponent  ci,
    short                    inputStd,
    Rect                     *activeSrcRect );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

VDGetBlackLevelValue

inputStd

A short integer that specifies the input video signal associated with this maximum source rectangle.

activeSrcRect

A pointer to a Rect (IV–2399) structure that is to receive the size and location information for the active source rectangle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Setting Source Characteristics” (V–2843)

VDGetBlackLevelValue

Returns the current black level value.

```
VideoDigitizerError VDGetBlackLevelValue (  
    VideoDigitizerComponent    ci,  
    unsigned short              *blackLevel );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

blackLevel

A pointer to an integer field that is to receive the current black level value. Black level values range from 0 to 65,535, where 0 represents the maximum black value and 65,535 represents the minimum black value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding set function, see `VDSetBlackLevelValue` (III–2031).

VDGetBrightness

Returns the current brightness value.

```
VideoDigitizerError VDGetBrightness (
    VideoDigitizerComponent  ci,
    unsigned short           *brightness );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

brightness

A pointer to an integer field that is to receive the current brightness value. Brightness values range from 0 to 65,535, where 0 is the darkest possible setting and 65,535 is the lightest possible setting.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding set function, see `VDSetBrightness` (III–2032).

VDGetClipState

Determines whether clipping is enabled.

```
VideoDigitizerError VDGetClipState (
    VideoDigitizerComponent  ci,
    short                    *clipEnable );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

VDGetCLUTInUse

`clipEnable`

A pointer to a short integer field that is to receive a value indicating whether clipping is enabled. The video digitizer component places 0 into the field if clipping is disabled, and 1 if it is enabled.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Video Clipping” (V–2853)

Related Java Methods

`quicktime.std.sg.VideoDigitizer.getClipState()`

See Also

For the corresponding set function, see `VDSetClipState` (III–2033).

VDGetCLUTInUse

Obtains the **color lookup table** used by a video digitizer component.

```
VideoDigitizerError VDGetCLUTInUse (
    VideoDigitizerComponent  ci,
    CTabHandle                *colorTableHandle );
```

`ci`

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`colorTableHandle`

A pointer to a field that is to receive a handle to a `ColorTable` (IV–2210) structure. The video digitizer component returns a handle to its **color lookup table**. Applications can then set the destination to use this returned `ColorTable` structure. Your application is responsible for disposing of this handle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Color” (V–2849)

VDGetCompressionTime

Confirms or quantifies a video digitizer's compression settings.

```
VideoDigitizerError VDGetCompressionTime (
    VideoDigitizerComponent    ci,
    OSType                     compressionType,
    short                      depth,
    Rect                       *srcRect,
    CodecQ                     *spatialQuality,
    CodecQ                     *temporalQuality,
    unsigned long              *compressTime );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

compressionType

A compressor type. This value corresponds to the component subtype of the compressor component. See “Codec Identifiers” (IV–2655).

depth

The depth at which the image is to be compressed. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 indicate 1-bit, 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

srcRect

A pointer to a `Rect` (IV–2399) structure that defines the portion of the source image to compress.

spatialQuality

A pointer to a field containing the desired compressed image quality (see below). The compressor sets this field to the closest actual quality that it can achieve. A value of `NIL` indicates that the client does not want this information.

temporalQuality

A pointer to a field containing the desired sequence temporal quality (see below). The compressor sets this field to the closest actual quality that it can achieve. A value of `NIL` indicates that the client does not want this information.

compressTime

A pointer to a field to receive the compression time, in milliseconds. Your component should return a long integer indicating the maximum number of milliseconds it would require to compress the specified image. If your component cannot determine the amount of time required to compress the

VDGetCompressionTypes

image, set this field to 0. A value of `NIL` indicates that the client does not want this information.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

spatialQuality and temporalQuality Constants

`codecMinQuality`

The minimum valid value for a CodecQ field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a CodecQ field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

The sequence grabber’s video compression settings dialog box uses this function to snap the quality slider to the correct value when working with a compression type that is specified by the video digitizer.

Version Notes

In QuickTime 1.5, video digitizers could provide compressed data directly to clients; however, there was no way to preflight the settings for compression. In QuickTime 2.1, this function was added to allow the video digitizer to quantify the compression time for the actual quality levels that will be used.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Compressed Source Devices” (V–2846)

VDGetCompressionTypes

Determines the image-compression capabilities of the video digitizer.

`VideoDigitizerError` `VDGetCompressionTypes` (


```
VideoDigitizerComponent    ci,  
VDCompressionListHandle   h );
```

ci

Identifies an application's connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

h

A handle to receive the compression information in one or more `VDCompressionList` (IV–2497) structures. If the digitizer supports more than one compression type, it creates an array of structures in this handle. The video digitizer returns information about its capabilities by formatting these structures.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

There must be a decompressor component of the appropriate type available in the system if an application is to display images from a compressed image sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Compressed Source Devices” (V–2846)

VDGetContrast

Returns the current contrast value.

```
VideoDigitizerError VDGetContrast (  
    VideoDigitizerComponent    ci,  
    unsigned short              *contrast );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

contrast

A pointer to an integer field that is to receive the current contrast value. The contrast value ranges from 0 to 65,535, where 0 represents no change to the basic image and larger values increase the contrast of the video image (they increase the slope of the transform).

VDGetCurrentFlags

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding set function, see VDSaveContrast (III–2036).

VDGetCurrentFlags

Returns status information about a specified video digitizer component.

```
VideoDigitizerError VDSaveContrast (  
    VideoDigitizerComponent    ci,  
    long                        *inputCurrentFlag,  
    long                        *outputCurrentFlag );
```

ci

The video digitizer component for the request. Applications obtain this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

inputCurrentFlag

A pointer to a long integer that is to receive the current input state flags for the video digitizer component; see “Video Digitizer Capabilities” (IV–2696).

outputCurrentFlag

A pointer to a long integer that is to receive the current output state flags for the video digitizer component; see “Video Digitizer Capabilities” (IV–2696).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

This function is often more convenient than VDSaveDigitizerInfo (III–1998). For example, this function provides a simple mechanism for determining whether a video digitizer is receiving a valid input signal. An application can retrieve the current input state flags and test the high-order bit by examining the sign of the returned value. If the value is negative (that is, the high-order bit, digitInSignalLock, is set to 1), the digitizer component is receiving a valid input signal.

Special Considerations

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Getting Information About Video Digitizer Components” (V–2843)

See Also

You can also use `VDGetDigitizerInfo` (III–1998) in your application to retrieve capability and current status information about a video digitizer component.

VDGetDataRate

Retrieves information that describes the performance capabilities of a video digitizer.

```
VideoDigitizerError VDGetDataRate (
    VideoDigitizerComponent  ci,
    long                     *milliSecPerFrame,
    Fixed                    *framesPerSecond,
    long                     *bytesPerSecond );
```

`ci`

Identifies the application’s connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`milliSecPerFrame`

A pointer to a long integer. The video digitizer returns a value that indicates the number of milliseconds of synchronous overhead involved in digitizing a single frame. This value includes the average delay incurred between the time when the digitizer requests a frame from its associated device, and the time at which the device delivers the frame.

`framesPerSecond`

A pointer to a fixed value. The video digitizer supplies the maximum rate at which it can capture video. Note that this value may differ from the rate that the application set with `VDSetFrameRate` (III–2041).

`bytesPerSecond`

A pointer to a long integer. Video digitizers that can return compressed image data return a value that indicates the approximate number of bytes per second

VDGetDigitizerInfo

that the digitizer is generating compressed data, given the current compression and frame rate settings.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Digitization” (V–2848)

See Also

For the corresponding set function, see `VDSetDataRate` (III–2037).

VDGetDigitizerInfo

Returns capability and status information about a specified video digitizer component.

```
VideoDigitizerError VDGetDigitizerInfo (  
    VideoDigitizerComponent    ci,  
    DigitizerInfo              *info );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

info

A pointer to a `DigitizerInfo` (IV–2234) structure. The function returns information describing the capabilities of the specified video digitizer into this structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Getting Information About Video Digitizer Components” (V–2843)

Related Java Methods

`quicktime.std.sg.VideoDigitizer.getDigitizerInfo()`

VDGetDigitizerRect

Returns the current digitizer rectangle.

```
VideoDigitizerError VDGetDigitizerRect (
    VideoDigitizerComponent    ci,
    Rect                       *digitizerRect );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

digitizerRect

A pointer to a `Rect` (IV–2399) structure that is to receive the size and location information for the current digitizer rectangle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Setting Source Characteristics” (V–2843)

See Also

For the corresponding set function, see `VDSetDigitizerRect` (III–2038).

VDGetDMADepths

Determines which pixel depths a digitizer supports.

```
VideoDigitizerError VDGetDMADepths (
    VideoDigitizerComponent    ci,
    long                       *depthArray,
    long                       *preferredDepth );
```

VDGetDMADepths

`ci`

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131).

`depthArray`

A pointer to a long integer. The video digitizer returns a value that indicates the depths it can support. Each depth is represented by a single bit in this field. More than one bit may be set to 1.

`preferredDepth`

A pointer to a long integer. Video digitizers that have a preferred depth value return that value in this field, using one of the possible values of the `depthArray` parameter. Digitizers that do not prefer any given value set this field to 0.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

The flags returned by this function augment the information that an application can obtain from the digitizer's output capability flags in the `DigitizerInfo` (IV-2234) structure. If a digitizer does not support this function but does support DMA, an application may assume that the digitizer can handle offscreen buffers at all of the depths indicated in its output capabilities flags. Applications may use the following enumerators to set bits in the field referred to by the `depthArray` parameter.

```
enum {  
    dmaDepth1      = 1      /* supports black and white */  
    dmaDepth2      = 2      /* supports 2-bit color */  
    dmaDepth4      = 4      /* supports 4-bit color */  
    dmaDepth8      = 8      /* supports 8-bit color */  
    dmaDepth16     = 16     /* supports 16-bit color */  
    dmaDepth32     = 32     /* supports 32-bit color */  
    dmaDepth2Gray  = 64     /* supports 2-bit grayscale */  
    dmaDepth4Gray  = 128    /* supports 4-bit grayscale */  
    dmaDepth8Gray  = 256    /* supports 8-bit grayscale */  
};
```

Special Considerations

Before a program that uses a video digitizer creates an offscreen buffer, it should call this function to determine the pixel depths supported by the digitizer. If possible, the program should use the preferred depth, in order to obtain the best possible display performance.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Color” (V–2849)

VDGetFieldPreference

Determines which field is being used in cases where the image is vertically scaled to half its original size.

```
VideoDigitizerError VDGetFieldPreference (
    VideoDigitizerComponent ci,
    short *fieldFlag );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

fieldFlag

Points to a field that is to receive a value (see below) indicating which field is being used.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

fieldFlag Constants

`vdUseAnyField`

Digitizer component decides which field to use.

`vdUseOddField`

Digitizer component uses odd field.

`vdUseEvenField`

Digitizer component uses even field.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Video Digitizer Utilities” (V–2853)

See Also

For the corresponding set function, see `VDSetFieldPreference` (III–2040).



VDGetHue

VDGetHue

Returns the current hue value.

```
VideoDigitizerError VDGetHue (
    VideoDigitizerComponent    ci,
    unsigned short             *hue );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

hue

A pointer to an integer that is to receive the current hue value. Hue is similar to the tint control on a television, and it is specified in degrees with complementary colors set 180 degrees apart (red is 0 degrees, green is +120 degrees, and blue is –120 degrees). Video digitizer components support hue values that range from 0 (–180 degrees shift in hue) to 65,535 (+179 degrees shift in hue), where 32,767 represents a 0 degree shift in hue.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding set function, see `VDSetHue` (III–2041).

VDGetImageDescription

Retrieves an `ImageDescription` (IV–2285) structure from a video digitizer.

```
VideoDigitizerError VDGetImageDescription (
    VideoDigitizerComponent    ci,
    ImageDescriptionHandle      desc );
```

ci

Identifies the application’s connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

desc

A handle. The video digitizer fills this handle with an `ImageDescription` (IV–2285) structure containing information about the digitizer’s current compression settings. The digitizer resizes the handle appropriately. It is the application’s responsibility to dispose of this handle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Compressed Source Devices” (V–2846)

VDGetInput

Returns data that identifies the currently active input video source.

```
VideoDigitizerError VDGetInput (
    VideoDigitizerComponent ci,
    short *input );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

input

A pointer to a short integer that is to receive the identifier for the currently active input video source. Video digitizer components number video sources sequentially, starting at 0. So, if the first source is active, this function sets the field referred to by the `input` parameter to 0.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selecting an Input Source” (V–2844)

VDGetInputColorSpaceMode

Related Java Methods

`quicktime.std.sg.VideoDigitizer.getInput()`

See Also

For the corresponding set function, see `VDSetInput` (III–2042).

VDGetInputColorSpaceMode

Determines whether a digitizer is operating in color or grayscale mode.

```
VideoDigitizerError VDGetInputColorSpaceMode (  
    VideoDigitizerComponent ci,  
    short *colorSpaceMode );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

colorSpaceMode

A pointer to a value that indicates whether the digitizer is operating in color (1) or grayscale (0) mode.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Applications can determine whether a digitizer component supports grayscale or color digitization by examining the digitizer component’s input capability flags. Specifically, if the `digiInDoesColor` flag is set to 1, the digitizer component supports color digitization. Similarly, if the `digiInDoesBW` flag is set to 1, the digitizer component supports grayscale digitization. Applications can use `VDGetCurrentFlags` (III–1996) to obtain the input capability flags of a digitizer component.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Color” (V–2849)

See Also

For the corresponding set function, see `VDSetInputColorSpaceMode` (III–2043).

VDGetInputFormat

Determines the format of the video signal provided by a specified video input source.

```
VideoDigitizerError VDGetInputFormat (
    VideoDigitizerComponent    ci,
    short                      input,
    short                      *format );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

input

The input video source for this request. Video digitizer components number video sources sequentially, starting at 0. So, to request information about the first video source, an application sets this parameter to 0. Applications can get the number of video sources supported by a video digitizer component by calling `VDGetNumberOfInputs` (III–2013).

format

A pointer to a short integer that is to receive a constant (see below) that specifies the video format of the specified input source.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

format Constants

`compositeIn`

The input video signal is in composite format.

`sVideoIn`

The input video signal is in s-video format.

`rgbComponentIn`

The input video signal is in RGB component format.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selecting an Input Source” (V–2844)

VDGetInputGammaRecord

VDGetInputGammaRecord

Retrieves a pointer to the active input VGammaRecord (IV–2498) structure for a video digitizer.

```
VideoDigitizerError VDGetInputGammaRecord (  
    VideoDigitizerComponent    ci,  
    VGammaRecPtr               *inputGammaPtr );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

inputGammaPtr

A pointer to a field that is to receive a pointer to an input VGammaRecord (IV–2498) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Gamma structures give applications complete control over color filtering transforms and are therefore more precise than the gamma values that can be set by calling `VDSetInputGammaValue` (III–2044).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding set function, see `VDSetInputGammaRecord` (III–2044).

VDGetInputGammaValue

Returns the current gamma values.

```
VideoDigitizerError VDGetInputGammaValue (  
    VideoDigitizerComponent    ci,  
    Fixed                      *channel1,  
    Fixed                      *channel2,  
    Fixed                      *channel3 );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

channel1

A pointer to a fixed integer field that is to receive the gamma value for the red component of the input video signal.

channel2

A pointer to a fixed integer field that is to receive the gamma value for the green component of the input video signal.

channel3

A pointer to a fixed integer field that is to receive the gamma value for the blue component of the input video signal.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding set function, see `VDSetInputGammaValue` (III–2044).

VDGetInputName

Gets the name of a video input.

```
VideoDigitizerError VDGetInputName (
    VideoDigitizerComponent  ci,
    long                      videoInput,
    Str255                    name );
```

ci

Specifies the video digitizer component for this operation. Applications can obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

videoInput

The input video source for this request. Video digitizer components number video sources sequentially, starting at 0. So, to request information about the first video source, an application sets this parameter to 0. Applications can get

VDGetKeyColor

the number of video sources supported by a video digitizer component by calling `VDGetNumberOfInputs` (III–2013).

name

The video input source’s name string.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

VDGetKeyColor

Obtains the index value of the active key color.

```
VideoDigitizerError VDGetKeyColor (  
    VideoDigitizerComponent    ci,  
    long                       *index );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

index

A pointer to a field that is to receive the index of the key color. This index value identifies the key color within the currently active **color lookup table**. If there are several active key colors, the video digitizer returns the first color from the key color list. Subsequently, applications use `VDGetNextKeyColor` (III–2013) to obtain other colors from the list. If there is no active key color, the function sets the field to –1.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Special Considerations

All video digitizer components that support key colors must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selectively Displaying Video” (V–2852)

See Also

For the corresponding set function, see `VDSetKeyColor` (III–2046).

VDGetKeyColorRange

Obtains the currently defined key color range.

```
VideoDigitizerError VDGetKeyColorRange (
    VideoDigitizerComponent ci,
    RGBColor *minRGB,
    RGBColor *maxRGB );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

minRGB

A pointer to a field that is to receive the lower bound of the key color range. The video digitizer component places the `RGBColor` (IV–2403) structure that corresponds to the lower end of the range in the field referred to by this parameter.

maxRGB

A pointer to a field that is to receive the upper bound of the key color range. The video digitizer component places the `RGBColor` (IV–2403) structure that corresponds to the upper end of the range in the field referred to by this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selectively Displaying Video” (V–2852)

See Also

For the corresponding set function, see `VDSetKeyColorRange` (III–2046).

VDGetMaskandValue

Obtains the appropriate alpha channel or blend mask value for a desired level of video blending.

VDGetMaskandValue

```
VideoDigitizerError VDGetMaskandValue (
    VideoDigitizerComponent ci,
    unsigned short          blendLevel,
    long                    *mask,
    long                    *value );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

blendLevel

The desired blend level. Valid values range from 0 to 65,535, where 0 corresponds to no video and 65,535 corresponds to all video.

mask

A pointer to a field that is to receive a value indicating which bits are meaningful in the data returned for the *value* parameter. The video digitizer component sets to 1 the bits that correspond to meaningful bits in the data returned for the *value* parameter.

value

A pointer to a field that is to receive data that can be used to obtain the desired blend level. The data returned for the *mask* parameter indicates which bits are valid in the data returned for this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The information returned by the digitizer component differs based on the type of blending supported by the component. In all cases, however, the returned value of the *value* parameter contains the value for the desired blend level, and the returned value of the *mask* parameter indicates which bits in the *value* parameter are meaningful. Bits in the returned *mask* parameter value that are set to 1 correspond to meaningful bits in the returned *value* parameter value.

For example, if an application requests a 50 percent video blend level from a digitizer that supports 8-bit alpha channels, the digitizer component might return `0xFF000000` in the *mask* parameter, identifying a full upper byte as the alpha channel, and `0x80000000` in the *value* parameter, specifying a 50 percent blend level.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selectively Displaying Video” (V–2852)

VDGetMaskPixMap

Retrieves the pixel map data for a component's blend mask.

```
VideoDigitizerError VDGetMaskPixMap (
    VideoDigitizerComponent    ci,
    PixMapHandle                maskPixMap );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

maskPixMap

A handle to a `PixMap` (IV–2332) structure. The video digitizer component returns the pixel map data for its blend mask into the `PixMap` (IV–2332) structure specified by this parameter. The video digitizer component resizes the handle as appropriate. Your application is responsible for disposing of this handle.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is supported only by digitizer components that support blend masks.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selectively Displaying Video” (V–2852)

VDGetMaxAuxBuffer

Obtains access to buffers that are located on special hardware.

```
VideoDigitizerError VDGetMaxAuxBuffer (
    VideoDigitizerComponent    ci,
    PixMapHandle                *pm,
    Rect                        *r );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

VDGetMaxSrcRect

pm

A pointer to a handle to a `PixelFormat` (IV-2332) structure. The video digitizer component returns a handle to the destination `PixelFormat` (IV-2332) structure in the field referred to by this parameter. Do not dispose of this structure. If the digitizer component cannot allocate a buffer, this handle is set to `NIL`.

r

A pointer to a `Rect` (IV-2399) structure. The video digitizer component places the coordinates of the largest output rectangle it can support into the structure referred to by this parameter.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Setting Video Destinations” (V-2845)

VDGetMaxSrcRect

Returns the maximum source rectangle.

```
VideoDigitizerError VDGetMaxSrcRect (  
    VideoDigitizerComponent    ci,  
    short                      inputStd,  
    Rect                       *maxSrcRect );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II-1130) or `OpenDefaultComponent` (II-1131).

inputStd

A short integer that specifies the input video signal associated with this maximum source rectangle.

maxSrcRect

A pointer to a `Rect` (IV-2399) structure that is to receive the size and location information for the maximum source rectangle.

function result See “Error Codes” (IV-2663). Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Setting Source Characteristics” (V–2843)

VDGetNextKeyColor

Obtains the index value of the active key colors in cases where the digitizer component supports multiple key colors.

```
VideoDigitizerError VDGetNextKeyColor (
    VideoDigitizerComponent  ci,
    long                     index );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

index

A field that is to receive the index of the next key color. This index value identifies the key color within the currently active **color lookup table**. If there are no more colors left in the list, the digitizer component sets the field referred to by the `index` parameter to `–1`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

All video digitizer components that support multiple key colors must support this function

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selectively Displaying Video” (V–2852)

VDGetNumberOfInputs

Returns the number of input video sources that a video digitizer component supports.

VDGetPlayThruDestination

```
VideoDigitizerError VDGetNumberOfInputs (
    VideoDigitizerComponent    ci,
    short                      *inputs );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

inputs

A pointer to an integer that is to receive the number of input video sources supported by the specified component. Video digitizer components number video sources sequentially, starting at 0. So, if a digitizer component supports two inputs, this function sets the field referred to by the `inputs` parameter to 1.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selecting an Input Source” (V–2844)

Related Java Methods

`quicktime.std.sg.VideoDigitizer.getNumberOfInputs()`

VDGetPlayThruDestination

Obtains information about the current video destination.

```
VideoDigitizerError VDGetPlayThruDestination (
    VideoDigitizerComponent    ci,
    PixMapHandle                *dest,
    Rect                        *destRect,
    MatrixRecord                 *m,
    RgnHandle                    *mask );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

dest

A pointer to a handle to a `PixelFormat` (IV-2332) structure. The video digitizer component returns a handle to the destination `PixelFormat` (IV-2332) structure in the field referred to by this parameter. It is the caller's responsibility to dispose of the `PixelFormat` structure.

destRect

A pointer to a `Rect` (IV-2399) structure. The video digitizer component places the coordinates of the output rectangle into the structure referred to by this parameter. If there is no output rectangle defined, the component returns an empty rectangle.

m

A pointer to a `MatrixRecord` (IV-2304) structure. The video digitizer component places the transformation matrix into the structure referred to by this parameter.

mask

A pointer to a handle to a `MacRegion` (IV-2303) structure. The video digitizer component places a handle to the mask region into the field referred to by this parameter. Applications can use masks to control the video into the destination rectangle. If there is no mask region defined, the digitizer component sets this returned handle to `NIL`. The caller is responsible for disposing of the `MacRegion` structure.

function result See "Error Codes" (IV-2663). Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: "Setting Video Destinations" (V-2845)

See Also

For the corresponding set function, see `VDSetPlayThruDestination` (III-2048).

VDGetPLIFilterType

Determines which phase locked loop (PLL) mode is currently active for a video digitizer.

VDGetPreferredImageDimensions

```
VideoDigitizerError VDGetPLLFilterType (  
    VideoDigitizerComponent    ci,  
    short                      *pllType );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

pllType

Points to a field that is to receive a value indicating which PLL mode is active. Values are 0 for broadcast mode and 1 for videotape recorder mode.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Video Digitizer Utilities” (V–2853)

See Also

For the corresponding set function, see `VDSetPLLFilterType` (III–2051).

VDGetPreferredImageDimensions

Gets the preferred image dimensions for a video digitizer.

```
VideoDigitizerError VDGetPreferredImageDimensions (  
    VideoDigitizerComponent    ci,  
    long                       *width,  
    long                       *height );
```

ci

Specifies the video digitizer component for this operation. Applications can obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

width

A pointer to the preferred image width.

height

A pointer to the preferred image height.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding set function, see `VDSetPreferredImageDimensions` (III–2051).

VDGetPreferredTimeScale

Determines a digitizer’s preferred time scale.

```
VideoDigitizerError VDGetPreferredTimeScale (
    VideoDigitizerComponent ci,
    TimeScale               *preferred );
```

ci

Identifies the application’s connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

preferred

A pointer to a time scale. The video digitizer returns information about its preferred time scale in this structure.

function result If the digitizer does not have a preferred time scale, it returns a result code of `digiUnimpErr`. See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Video Digitizer Utilities” (V–2853)

VDGetSaturation

Returns the current saturation value.

```
VideoDigitizerError VDGetSaturation (
    VideoDigitizerComponent ci,
    unsigned short          *saturation );
```

VDGetSharpness

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

saturation

A pointer to an integer that is to receive the current saturation value. The saturation value controls color intensity. For example, at high saturation levels, red appears to be red; at low saturation, red appears as pink. Valid saturation values range from 0 to 65,535, where 0 is the minimum saturation value and 65,535 specifies maximum saturation.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding set function, see `VDSetSaturation` (III–2053).

VDGetSharpness

Returns the current sharpness value.

```
VideoDigitizerError VDGetSharpness (  
    VideoDigitizerComponent    ci,  
    unsigned short              *sharpness );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

sharpness

A pointer to an integer that is to receive the current sharpness value. The sharpness value ranges from 0 to 65,535, where 0 represents no sharpness filtering and 65,535 represents full sharpness filtering. Higher values result in a visual impression of increased picture sharpness.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`
 Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding set function, see `VDSetSharpness` (III–2053).

VDGetSoundInputDriver

Retrieves information about a video digitizer’s sound input driver.

```
VideoDigitizerError VDGetSoundInputDriver (
    VideoDigitizerComponent  ci,
    Str255                    soundDriverName );
```

ci

Identifies the application’s connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

soundDriverName

A pointer to a string. The video digitizer returns the name of its sound input driver. If the digitizer does not have an associated driver, it returns a result code of `digiUnimpErr`.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`
 Programming summary: “Video Digitizer Utilities” (V–2853)

VDGetSoundInputSource

Instructs your video digitizer component to return the sound input source associated with a particular video input.

```
VideoDigitizerError VDGetSoundInputSource (
    VideoDigitizerComponent  ci,
    long                      videoInput,
    long                      *soundInput );
```

VDGetTimeCode

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

videoInput

The input video source for this request. Video digitizer components number video sources sequentially, starting at 0. So, to request information about the first video source, an application sets this parameter to 0. Applications can get the number of video sources supported by a video digitizer component by calling `VDGetNumberOfInputs` (III–2013).

soundInput

The sound input index to use with the sound input driver returned by `VDGetSoundInputDriver` (III–2019).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Some video digitizers may associate different sound inputs with each video input. `VDGetSoundInputDriver` (III–2019) returns the name of the sound input driver that the sound input is associated with.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Compressed Source Devices” (V–2846)

VDGetTimeCode

Instructs your video digitizer component to return timecode information for the incoming video signal.

```
VideoDigitizerError VDGetTimeCode (
    VideoDigitizerComponent    ci,
    TimeRecord                  *atTime,
    void                        *timeCodeFormat,
    void                        *timeCodeTime );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

atTime

A pointer to a `TimeRecord` (IV–2486) structure to receive the QuickTime movie time value corresponding to the timecode information.

timeCodeFormat

A pointer to a `TimeCodeDef` (IV–2482) structure. Your video digitizer component returns the movie’s timecode definition information in this structure.

timeCodeTime

A pointer to a `TimeCodeRecord` (IV–2484) structure. Your video digitizer component returns the time value corresponding to the movie time contained in this structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

Typically, this function is called once, at the beginning of a capture session. The use of this function assumes that the timecoding for the entire capture session will be continuous.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Digitization” (V–2848)

VDGetVBlankRect

Returns the vertical blanking rectangle.

```
VideoDigitizerError VDGetVBlankRect (
    VideoDigitizerComponent  ci,
    short                    inputStd,
    Rect                      *vBlankRect );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

inputStd

A short integer (see below) that identifies the signaling standard used in the source video signal.

VDGetVideoDefaults

vBlankRect

A pointer to a Rect (IV–2399) structure that is to receive the size and location information for the vertical blanking rectangle.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inputStd Constants

ntscIn

Input video signal to digitize is in NTSC format.

palIn

Input video signal to digitize is in PAL format.

secamIn

Input video signal to digitize is in SECAM format.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Setting Source Characteristics” (V–2843)

VDGetVideoDefaults

Returns the recommended values for many of the analog video parameters that may be set by applications.

```
VideoDigitizerError VDGetVideoDefaults (
    VideoDigitizerComponent    ci,
    unsigned short              *blackLevel,
    unsigned short              *whiteLevel,
    unsigned short              *brightness,
    unsigned short              *hue,
    unsigned short              *saturation,
    unsigned short              *contrast,
    unsigned short              *sharpness );
```

ci

The video digitizer component for the request. Applications obtain this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

blackLevel

A pointer to an integer that is to receive the default black level value. Black level values range from 0 to 65,535, where 0 represents the maximum black value and 65,535 represents the minimum black value.

whiteLevel

A pointer to an integer that is to receive the default white level value. White level values range from 0 to 65,535, where 0 represents the minimum white value and 65,535 represents the maximum white value.

brightness

A pointer to an integer that is to receive the default brightness value. Brightness values range from 0 to 65,535, where 0 is the darkest possible setting and 65,535 is the lightest possible setting.

hue

A pointer to an integer that is to receive the default hue value. Hue is similar to the tint control on a television, and it is specified in degrees with complementary colors set 180 degrees apart (red is 0 degrees, green is +120 degrees, and blue is -120 degrees). Video digitizer components support hue values that range from 0 (-180 degrees shift in hue) to 65,535 (+179 degrees shift in hue), where 32,767 represents a 0 degree shift in hue.

saturation

A pointer to an integer that is to receive the default saturation value. The saturation value controls color intensity. For example, at high saturation levels, red appears to be red; at low saturation, red appears as pink. Valid saturation values range from 0 to 65,535, where 0 is the minimum saturation value and 65,535 specifies maximum saturation.

contrast

A pointer to an integer that is to receive the default contrast value. The contrast value ranges from 0 to 65,535, where 0 represents no change to the basic image and larger values increase the contrast of the video image (they increase the slope of the transform).

sharpness

A pointer to an integer that is to receive the default sharpness value. The sharpness value ranges from 0 to 65,535, where 0 represents no sharpness filtering and 65,535 represents full sharpness filtering. Higher values result in a visual impression of increased picture sharpness.

function result See “Error Codes” (IV-2663). Returns noErr if there is no error.

Discussion

All video digitizer components must support this function.

VDGetWhiteLevelValue

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

VDGetWhiteLevelValue

Returns the current white level value.

```
VideoDigitizerError VDGetWhiteLevelValue (
    VideoDigitizerComponent    ci,
    unsigned short              *whiteLevel );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

whiteLevel

A pointer to an integer that is to receive the current white level value. White level values range from 0 to 65,535, where 0 represents the minimum white value and 65,535 represents the maximum white value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding set function, see `VDSetWhiteLevelValue` (III–2055).

VDGrabOneFrame

Instructs the video digitizer component to digitize a single frame of source video.

```
VideoDigitizerError VDGrabOneFrame (
    VideoDigitizerComponent    ci );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If the specified digitizer component is already digitizing continuously when the application calls this function, the digitizer component returns the next digitized frame and then stops. If the digitizer component is stopped, the component digitizes a single frame and then stops. To resume continuous digitization, applications should call `VDSetPlayThruOnOff` (III–2050).

Special Considerations

This function supports synchronous single-frame video digitization; that is, the digitizer component does not return control to your application until it has successfully processed the next video frame. Some video digitizer components may also support asynchronous single-frame digitization. Applications can request asynchronous digitization by calling `VDGrabOneFrameAsync` (III–2025).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Digitization” (V–2848)

VDGrabOneFrameAsync

Instructs the video digitizer component to start to digitize asynchronously a single frame of source video.

```
VideoDigitizerError VDGrabOneFrameAsync (
    VideoDigitizerComponent ci,
    short buffer );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

buffer

Identifies the next output buffer. The value of this parameter must correspond to a valid index into the list of buffers that you supply when your application

VDPreFlightDestination

calls `VDSetupBuffers` (III–2055). Note that this value is zero-based (that is, you must set this parameter to 0 to refer to the first buffer in the buffer list).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

When calling this function, the application specifies the next destination video buffer, allowing the digitizer component to quickly switch from the current buffer to the next buffer. In this manner, your application’s ability to grab video at high frame rates is enhanced. If the specified digitizer component is already digitizing continuously when the application calls this function, the digitizer component returns the next digitized frame and then stops. If the digitizer component is stopped, the component digitizes a single frame and then stops. To resume continuous digitization, applications should call `VDSetPlayThruOnOff` (III–2050).

This function also allows applications to use more than one destination buffer for the digitized video. The application defines these buffers by calling `VDSetupBuffers` (III–2055). The application specifies one of these destination buffers for the digitized frame when it calls `VDSetPlayThruDestination` (III–2048) or `VDSetPlayThruGlobalRect` (III–2049).

Special Considerations

Applications can determine whether a video digitizer component supports asynchronous frame grabbing by using `VDGetCurrentFlags` (III–1996) to retrieve the digitizer component’s output capability flags. If the `digiOutDoesAsyncGrabs` flag is set to 1, the digitizer component supports both this function and `VDDone` (III–1988). If a video digitizer component does not support asynchronous digitization, applications must use `VDGrabOneFrame` (III–2024) to perform single-frame digitization.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Digitization” (V–2848)

VDPreFlightDestination

Verifies that a video digitizer component can support a set of destination settings intended for use with `VDSetPlayThruDestination` (III–2048).

```
VideoDigitizerError VDPreflightDestination (  
    VideoDigitizerComponent    ci,
```



```
Rect          *digitizerRect,
PixMap        **dest,
RectPtr       destRect,
MatrixRecordPtr m );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

digitizerRect

A pointer to a `Rect` (IV–2399) structure that contains the size and location information for the digitizer rectangle. The coordinates of this rectangle must be relative to the maximum source rectangle. In addition, the digitizer rectangle must be within the maximum source rectangle. For a discussion of the relationship between these rectangles, see “Video Digitizer Components” in *Inside Macintosh: QuickTime Components* (listed in the **bibliography**). If the video digitizer component cannot accommodate the specified rectangle, it changes the coordinates in this structure to specify a rectangle that it can support and sets the result to `qtParamErr`.

dest

A handle to the destination `PixMap` (IV–2332) structure.

destRect

A pointer to a `Rect` (IV–2399) structure that specifies the size and location of the video destination. This is an optional parameter. Applications may specify a transformation matrix to control the placement and scaling of the video image in the destination `PixMap` (IV–2332) structure. In this case, the `destRect` parameter is set to `NIL` and the `m` parameter specifies the matrix. The destination rectangle must be in the coordinate system of the destination `PixMap` structure specified by the `dest` parameter. If the video digitizer component cannot accommodate this rectangle, it changes the coordinates in the structure to specify a rectangle that it can support and sets the result to `qtParamErr`.

m

A pointer to a `MatrixRecord` (IV–2304) structure containing the transformation matrix for the destination video image. This is an optional parameter. Applications may specify a destination rectangle to control the placement and scaling of the video image in the destination `PixMap` (IV–2332) structure. In this case, the `m` parameter is set to `NIL` and the `destRect` parameter specifies the destination rectangle. If the `destRect` parameter is `NIL`, you can determine the destination rectangle for simple matrices by calling `TransformRect` (III–1954) using the current digitizer rectangle and this matrix. If the video digitizer component cannot accommodate this matrix, it changes the values in the

VDPreflightGlobalRect

structure to define a matrix that it can support and sets the result to `qtParamErr`. Applications can determine the capabilities of a video digitizer component by calling `VDGetDigitizerInfo` (III–1998).

function result The application provides the desired settings as parameters to this function. The video digitizer component then examines those settings. If the digitizer component can support the specified settings, it sets the result code to `noErr`. If the digitizer component cannot support the settings, it alters the input settings to reflect values that it can support and returns a result code of `qtParamErr`. See “Error Codes” (IV–2663).

Discussion

All video digitizer components must support this function. Applications should use this function to test destination settings whenever a video digitizer component cannot support arbitrary scaling.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Setting Video Destinations” (V–2845)

See Also

Applications can determine the capabilities of a video digitizer component by examining the output capability flags, using `VDGetCurrentFlags` (III–1996). Specifically, if the `digiOutDoesStretch` and `digiOutDoesShrink` flags are set to 1 in the output capability flag, the digitizer component supports arbitrary scaling.

VDPreflightGlobalRect

Verifies that a video digitizer component can support a set of destination settings intended for use with `VDSetPlayThruGlobalRect` (III–2049).

```
VideoDigitizerError VDPreflightGlobalRect (
    VideoDigitizerComponent    ci,
    GrafPtr                    theWindow,
    Rect                        *globalRect );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

theWindow

A pointer to the destination window.

globalRect

A pointer to a Rect (IV–2399) structure that specifies the size and location of the video destination. This rectangle must be in the coordinate system of the destination window specified by the *theWindow* parameter. If the video digitizer component cannot accommodate this rectangle, it changes the coordinates in the structure to specify a rectangle that it can support and sets the result to *qtParamErr*.

function result Returns *qtParamErr* if the video digitizer component cannot accommodate the destination rectangle. Returns *digiUnimpErr* if the video digitizer component does not support placing destination video into a rectangle that crosses screens. See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

Discussion

Applications should use this function to determine whether a video digitizer supports placing destination video into a rectangle that crosses screens. Digitizers that do not support this capability return a result of *digiUnimpErr*.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: *QuickTimeComponents.h*

Programming summary: “Setting Video Destinations” (V–2845)

See Also

See *VDPreflightDestination* (III–2026).

VDReleaseAsyncBuffers

Releases the buffers that were allocated with *VDSetupBuffers* (III–2055).

```
VideoDigitizerError VDReleaseAsyncBuffers (
    VideoDigitizerComponent    ci );
```

ci

The video digitizer component for the request. Applications obtain this reference from *OpenComponent* (II–1130) or *OpenDefaultComponent* (II–1131).

function result See “Error Codes” (IV–2663). Returns *noErr* if there is no error.

VDReleaseCompressBuffer

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Controlling Digitization” (V–2848)

VDReleaseCompressBuffer

Frees a buffer received from VDCompressDone (III–1986).

```
VideoDigitizerError VDReleaseCompressBuffer (
    VideoDigitizerComponent    ci,
    Ptr                        bufferAddr );
```

ci

Identifies the application’s connection to the video digitizer component. An application obtains this value from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

bufferAddr

Points to the location of the buffer to be released. This address must correspond to a buffer address that the application obtained from VDCompressDone (III–1986).

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Controlling Compressed Source Devices” (V–2846)

VDResetCompressSequence

Forces the video digitizer to insert a **key frame** into a temporally compressed image sequence.

```
VideoDigitizerError VDResetCompressSequence (
    VideoDigitizerComponent    ci );
```

ci

Identifies the application’s connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Compressed Source Devices” (V–2846)

V

VDSetsBlackLevelValue

Sets the current black level value.

```
VideoDigitizerError VDSetsBlackLevelValue (
    VideoDigitizerComponent ci,
    unsigned short          *blackLevel );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

blackLevel

A pointer to an integer that contains the new black level value. Black level values range from 0 to 65,535, where 0 represents the maximum black value and 65,535 represents the minimum black value. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding get function, see `VDGetBlackLevelValue` (III–1990).

VDSetBrightness

VDSetBrightness

Sets the current brightness value.

```
VideoDigitizerError VDSetsBrightness (
    VideoDigitizerComponent  ci,
    unsigned short           *brightness );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

brightness

A pointer to an integer that contains the new brightness value. Brightness values range from 0 to 65,535, where 0 is the darkest possible setting and 65,535 is the lightest possible setting. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding get function, see `VDGetBrightness` (III–1991).

VDSetClipRgn

Defines a clipping region for a video digitizer.

```
VideoDigitizerError VDSetsClipRgn (
    VideoDigitizerComponent  ci,
    RgnHandle                 clipRegion );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

clipRegion

A handle to a `MacRegion` (IV–2303) structure that defines the clipping region.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Video Clipping” (V–2853)

Related Java Methods

quicktime.std.sg.VideoDigitizer.setClipRgn()

VDSetClipState

Controls whether clipping is enabled.

```
VideoDigitizerError VDSetClipState (
    VideoDigitizerComponent ci,
    short clipEnable );
```

ci

The video digitizer component for the request. Applications obtain this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

clipEnable

Controls whether clipping is enabled. Place 0 into the short integer if clipping is disabled, and 1 if it is enabled.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Video Clipping” (V–2853)

Related Java Methods

quicktime.std.sg.VideoDigitizer.setClipState()

See Also

For the corresponding get function, see VDGetClipState (III–1991).

VDSetCompression

Specifies certain compression parameters.

```
VideoDigitizerError VDSetCompression (
    VideoDigitizerComponent    ci,
    OSType                     compressType,
    short                       depth,
    Rect                        *bounds,
    CodecQ                     spatialQuality,
    CodecQ                     temporalQuality,
    long                       keyFrameRate );
```

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

compressType

A compressor type. This value corresponds to the component subtype of the compressor component; see “Codec Identifiers” (IV–2655).

depth

The depth at which the image is likely to be viewed. Compressors may use this as an indication of the color or grayscale resolution of the image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 correspond to 1-bit, 2-bit, 4-bit, and 8-bit grayscale images.

bounds

A pointer to a `Rect` (IV–2399) structure that defines the desired boundaries of the compressed image.

spatialQuality

A constant (see below) that indicates the desired image quality for each frame in the sequence.

temporalQuality

A constant (see below) that indicates the desired temporal quality for the sequence as a whole.

keyFrameRate

The maximum number of frames to allow between **key frames**. This value defines the minimum rate at which key frames are to appear in the compressed sequence; however, the video digitizer may insert key frames more often than an application specifies. If the application requests no temporal compression

(that is, the application set the `temporalQuality` parameter to 0), the video digitizer ignores this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

spatialQuality and temporalQuality Constants

`codecMinQuality`

The minimum valid value for a `CodecQ` field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a `CodecQ` field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Compressed Source Devices” (V–2846)

VDSetsCompressionOnOff

Allows an application to start and stop compression by video digitizers that can deliver either compressed or uncompressed image data.

```

VideoDigitizerError VDSetsCompressionOnOff (
    VideoDigitizerComponent    ci,
    Boolean                     state );

```

VDSetContrast

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

state

A Boolean value that indicates whether to enable or disable compression. Applications set this parameter to `TRUE` to enable compression. Setting it to `FALSE` disables compression.

function result Digitizers that only provide compressed data have their `digitOutDoesCompressOnly` flag set to 1, rather than 0. These digitizers may either ignore this function or return a nonzero result code. See “Error Codes” (IV–2663). Return `noErr` if there is no error.

Discussion

Applications must call this function before they call either `VDSetCompression` (III–2034) or `VDCompressOneFrameAsync` (III–1988). This allows the video digitizer to prepare for the operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Compressed Source Devices” (V–2846)

VDSetContrast

Sets the current contrast value.

```
VideoDigitizerError VDSetContrast (  
    VideoDigitizerComponent    ci,  
    unsigned short              *contrast );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

contrast

A pointer to an integer that contains the new contrast value. The contrast value ranges from 0 to 65,535, where 0 represents no change to the basic image and larger values increase the contrast of the video image (they increase the slope

of the transform). The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding get function, see `VDGetContrast` (III–1995).

VDSetDataRate

Instructs your video digitizer component to limit the rate at which it delivers compressed, digitized video data.

```
VideoDigitizerError VDSetDataRate (
    VideoDigitizerComponent ci,
    long bytesPerSecond );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

bytesPerSecond

The maximum data rate requested by the application, in bytes per second. This parameter is set to 0 to remove any data-rate restrictions.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This function is valid only for video digitizer components that can deliver compressed video; that is, components that support the `VDCompressOneFrameAsync` (III–1988) function. Components that support data-rate limiting set the `codecInfoDoesRateConstrain` flag to 1 in the `compressFlags` field of the `VDCompressionList` (IV–2497) structure returned by the component in response to the `VDGetCompressionTypes` (III–1994) function. Your video digitizer component should return this data-rate limit in the `bytesPerSecond` parameter of the existing `VDGetDataRate` (III–1997) function.

VDSetDestinationPort

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Compressed Source Devices” (V–2846)

See Also

For the corresponding get function, see `VDGetDataRate` (III–1997).

VDSetDestinationPort

Sets the destination port for a video digitizer.

```
VideoDigitizerError VDSetsDestinationPort (
    VideoDigitizerComponent  ci,
    CGrafPtr                 destPort );
```

ci

Specifies the video digitizer component for this operation. Applications can obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

destPort

A pointer to a `CGrafPort` (IV–2168) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `QuickTimeComponents.h`

VDSetDigitizerRect

Sets the current video digitizer rectangle.

```
VideoDigitizerError VDSetsDigitizerRect (
    VideoDigitizerComponent  ci,
    Rect                     *digitizerRect );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

digitizerRect

A pointer to a `Rect` (IV–2399) structure that contains the size and location information for the digitizer rectangle. The coordinates of this rectangle must be relative to the maximum source rectangle. In addition, the digitizer rectangle must be within the maximum source rectangle. For a discussion of the relationship between these rectangles, see “Video Digitizer Components” in *Inside Macintosh: QuickTime Components* (listed in the **bibliography**).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Setting Source Characteristics” (V–2843)

See Also

For the corresponding get function, see `VDGetDigitizerRect` (III–1999).

VDSetsDigitizerUserInterrupt

Sets custom interrupt functions.

```

VideoDigitizerError VDSetsDigitizerUserInterrupt (
    VideoDigitizerComponent  ci,
    long                      flags,
    VdigIntUPP                userInterruptProc,
    long                      refcon );

```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

flags

Indicates when the interrupt function is to be called. If bit 0 is set to 1, the video digitizer component calls the custom interrupt procedure each time it starts to display an even-line field. If bit 1 is set to 1, the video digitizer component calls

VDSetFieldPreference

the custom interrupt procedure each time it starts to display an odd-line field. Applications may set both bits to 1.

`userInterruptProc`

A **Universal Procedure Pointer** to a `VdigIntProc` (III–2154) callback.

Applications can set this parameter to `NIL` to remove a `VdigIntProc` callback.

`refcon`

Contains parameter data that is appropriate for the callback. Use this parameter to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Video Digitizer Utilities” (V–2853), “Video Digitizer Utilities” (V–2853)

VDSetFieldPreference

Specifies which field to use in cases where the vertical scaling is less than half size.

```
VideoDigitizerError VDSetFieldPreference (  
    VideoDigitizerComponent    ci,  
    short                      fieldFlag );
```

`ci`

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

`fieldFlag`

A constant (see below) that indicates which field to use.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

fieldFlag Constants

`vdUseAnyField`

Digitizer component decides which field to use.

`vdUseOddField`

Digitizer uses odd field.

`vdUseEvenField`

Digitizer uses even field.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Video Digitizer Utilities” (V–2853)

See Also

For the corresponding get function, see VDSetsFieldPreference (III–2001).

VDSetsFrameRate

Indicates an application’s desired frame rate to the video digitizer.

```
VideoDigitizerError VDSetsFrameRate (
    VideoDigitizerComponent ci,
    Fixed framesPerSecond );
```

ci

Identifies the application’s connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

framesPerSecond

The application’s desired frame rate. Applications may set this parameter to 0 to return the digitizer to its default frame rate (typically 29.97 frames per second).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Controlling Digitization” (V–2848)

VDSetsHue

Sets the current hue value.

```
VideoDigitizerError VDSetsHue (
```

VDSetInput

```
VideoDigitizerComponent    ci,  
unsigned short             *hue );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

hue

A pointer to an integer that contains the new hue value. Hue is similar to the tint control on a television, and it is specified in degrees with complementary colors set 180 degrees apart (red is 0 degrees, green is +120 degrees, and blue is –120 degrees). Video digitizer components support hue values that range from 0 (–180 degrees shift in hue) to 65,535 (+179 degrees shift in hue), where 32,767 represents a 0 degree shift in hue. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding get function, see `VDGetHue` (III–2002).

VDSetInput

Selects the input video source for a video digitizer component.

```
VideoDigitizerError VDSetInput (  
    VideoDigitizerComponent    ci,  
    short                     input );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

input

The input video source for this request. Video digitizer components number video sources sequentially, starting at 0. To request the first video source, an application sets this parameter to 0.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Selecting an Input Source” (V–2844)

Related Java Methods

quicktime.std.sg.VideoDigitizer.setInput()

See Also

For the corresponding get function, see VDGetInput (III–2003).

VDSetInputColorSpaceMode

Chooses between color and grayscale digitized video.

```
VideoDigitizerError VDSetInputColorSpaceMode (
    VideoDigitizerComponent ci,
    short                    colorSpaceMode );
```

ci

The video digitizer component for the request. Applications obtain this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

colorSpaceMode

Controls color digitization. Set to 0 for grayscale, 1 for color.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Controlling Color” (V–2849)

See Also

For the corresponding get function, see VDGetInputColorSpaceMode (III–2004).

VDSetInputGammaRecord

VDSetInputGammaRecord

Changes the active input gamma data structure.

```
VideoDigitizerError VDSetInputGammaRecord (  
    VideoDigitizerComponent    ci,  
    VDGamRecPtr                inputGammaPtr );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

inputGammaPtr

A `VDGammaRecord` (IV–2498) structure.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding get function, see `VDGetInputGammaRecord` (III–2006).

VDSetInputGammaValue

Sets the gamma values.

```
VideoDigitizerError VDSetInputGammaValue (  
    VideoDigitizerComponent    ci,  
    Fixed                      channel1,  
    Fixed                      channel2,  
    Fixed                      channel3 );
```

ci

The video digitizer component for the request. Applications obtain this reference from the Component Manager’s `OpenComponent` function.

channel1

The gamma value for the red component of the input video signal.

channel2

The gamma value for the green component of the input video signal.

channel3

The gamma value for the blue component of the input video signal.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Discussion

These gamma values control the brightness of the input video signal. Your application can implement special color effects, such as turning off specific color channels, by calling this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding get function, see VDSetsInputGammaValue (III–2006).

V

VDSetsInputStandard

Specifies the input signaling standard to digitize.

```
VideoDigitizerError VDSetsInputStandard (
    VideoDigitizerComponent ci,
    short inputStandard );
```

ci

The video digitizer component for the request. Applications obtain this reference from OpenComponent (II–1130) or OpenDefaultComponent (II–1131).

inputStandard

A short integer (see below) that identifies the input signaling standard.

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

inputStandard Constants

ntscIn

Input video signal to digitize is in NTSC format.

palIn

Input video signal to digitize is in PAL format.

secamIn

Input video signal to digitize is in SECAM format.

VDSetKeyColor

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selecting an Input Source” (V–2844)

VDSetKeyColor

Sets the key color for video digitizing.

```
VideoDigitizerError VDSetsKeyColor (
    VideoDigitizerComponent  ci,
    long                      index );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

index

The new key color. This value must correspond to a color in the current **color lookup table**.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

All video digitizer components that support key colors must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selectively Displaying Video” (V–2852)

See Also

For the corresponding get function, see `VDGetKeyColor` (III–2008).

VDSetKeyColorRange

Defines a key color range for video digitizing.

```

VideoDigitizerError VDSetsKeyColorRange (
    VideoDigitizerComponent    ci,
    RGBColor                   *minRGB,
    RGBColor                   *maxRGB );

```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

minRGB

A pointer to a field that contains the lower bound of the key color range. All colors in the color table between the color specified by the `minRGB` parameter and the color specified by the `maxRGB` parameter are considered key colors.

maxRGB

A pointer to a field that contains the upper bound of the key color range. All colors in the color table between the color specified by the `minRGB` parameter and the color specified by the `maxRGB` parameter are considered key colors.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selectively Displaying Video” (V–2852)

See Also

For the corresponding get function, see `VDGetKeyColorRange` (III–2009).

VDSetsMasterBlendLevel

Sets the blend level value for the input video signal.

```

VideoDigitizerError VDSetsMasterBlendLevel (
    VideoDigitizerComponent    ci,
    unsigned short             *blendLevel );

```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

blendLevel

A pointer to a field that specifies the new master blend level. Valid values range from 0 to 65,535, where 0 corresponds to no video and 65,535 corresponds to all

VDSetPlayThruDestination

video. The digitizer component returns the new value in this field, so your application can avoid using unsupported values in future requests.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Selectively Displaying Video” (V–2852)

VDSetPlayThruDestination

Establishes the destination settings for a video digitizer component.

```
VideoDigitizerError VDSetPlayThruDestination (
    VideoDigitizerComponent    ci,
    PixMapHandle                dest,
    RectPtr                     destRect,
    MatrixRecordPtr             m,
    RgnHandle                   mask );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

dest

A handle to the destination `PixMap` (IV–2332) structure. This pixel map may be in the video frame buffer of the Macintosh computer, or it may specify an offscreen buffer.

destRect

A pointer to a `Rect` (IV–2399) structure that specifies the size and location of the video destination. This rectangle must be in the coordinate system of the destination `PixMap` (IV–2332) structure specified by the *dest* parameter.

m

A pointer to a `MatrixRecord` (IV–2304) structure containing the transformation matrix for the destination video image. To determine the capabilities of a video digitizer component, you can call `VDGetDigitizerInfo` (III–1998) in your application.

mask

A handle to a `MacRegion` (IV–2303) structure that defines a mask. Applications can use masks to control clipping of the video into the destination rectangle. This mask region is defined in the destination coordinate space.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Special Considerations

Applications set the source digitizer rectangle by calling `VDSetDigitizerRect` (III–2038).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Setting Video Destinations” (V–2845), “Video Clipping” (V–2853)

See Also

For the corresponding get function, see `VDGetPlayThruDestination` (III–2014).

VDSetPlayThruGlobalRect

Establishes the destination settings for a video digitizer component that is to digitize into a global rectangle.

```
VideoDigitizerError VDSetPlayThruGlobalRect (
    VideoDigitizerComponent  ci,
    GrafPtr                  theWindow,
    Rect                      *globalRect );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

theWindow

A pointer to the destination window.

globalRect

A pointer to a `Rect` (IV–2399) structure that specifies the size and location of the video destination. This rectangle must be in the coordinate system of the destination window specified by the `theWindow` parameter.

VDSetPlayThruOnOff

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

The application provides the desired settings as parameters to this function. Not all video digitizer components support global rectangles.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Setting Video Destinations” (V–2845), “Video Clipping” (V–2853)

VDSetPlayThruOnOff

Controls continuous digitization.

```
VideoDigitizerError VDSetPlayThruOnOff (  
    VideoDigitizerComponent  ci,  
    short                    state );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

state

A short integer (see below) that specifies whether to use continuous digitization. When an application stops continuous digitization, the video digitizer component must restore its alpha channel, blending mask, or key color settings to graphics mode.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

state Constants

`digitizerOff`

Turns off continuous digitization.

`digitizerOn`

Turns on continuous digitization.

Discussion

When opened, video digitizer components are always set to off, so that no digitization is taking place. Your application can use this function to turn continuous digitization on and off.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Digitization” (V–2848)

See Also

Applications can request single-frame digitization by calling `VDGrabOneFrame` (III–2024) or `VDGrabOneFrameAsync` (III–2025).

VDSetsPLLFilterType

Specifies which phase locked loop (PLL) is to be active.

```
VideoDigitizerError VDSetsPLLFilterType (
    VideoDigitizerComponent  ci,
    short                    pllType );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

pllType

Indicates which PLL is to be active. Values are 0 for broadcast mode and 1 for videotape recorder mode.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Video Digitizer Utilities” (V–2853)

See Also

For the corresponding get function, see `VDGetPLLFilterType` (III–2015).

VDSetsPreferredImageDimensions

Sets the preferred image dimensions for a video digitizer.

```
VideoDigitizerError VDSetsPreferredImageDimensions (
    VideoDigitizerComponent  ci,
```



VDSetPreferredPacketSize

```
long          width,  
long          height );
```

ci

Specifies the video digitizer component for this operation. Applications can obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

width

The preferred image width.

height

The preferred image height.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

For the corresponding get function, see `VDGetPreferredImageDimensions` (III–2016).

VDSetPreferredPacketSize

Sets the preferred packet size for video digitizing.

```
VideoDigitizerError VDSetPreferredPacketSize (  
    VideoDigitizerComponent  ci,  
    long                      preferredPacketSizeInBytes );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

preferredPacketSizeInBytes

The preferred packet size in bytes.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

This function was added in QuickTime 2.5 to support videoconferencing applications.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Digitization” (V–2848)

VDSetSaturation

Sets the saturation value.

```
VideoDigitizerError VDSetSaturation (
    VideoDigitizerComponent ci,
    unsigned short          *saturation );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

saturation

A pointer to an integer that contains the new saturation value. The saturation value controls color intensity. For example, at high saturation levels, red appears to be red; at low saturation, red appears as pink. Valid saturation values range from 0 to 65,535, where 0 is the minimum saturation value and 65,535 specifies maximum saturation. The video digitizer component attempts to set the saturation value to the value specified by this parameter. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding get function, see `VDGetSaturation` (III–2017).

VDSetSharpness

Sets the sharpness value.

```
VideoDigitizerError VDSetSharpness (
```

VDSetTimeBase

```
VideoDigitizerComponent    ci,  
unsigned short              *sharpness );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

sharpness

A pointer to an integer that contains the new sharpness value. The sharpness value ranges from 0 to 65,535, where 0 represents no sharpness filtering and 65,535 represents full sharpness filtering. Higher values result in a visual impression of increased picture sharpness. The video digitizer component attempts to set the sharpness value to the value specified by this parameter. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding get function, see `VDGetSharpness` (III–2018).

VDSetTimeBase

Establishes the video digitizer’s time coordinate system.

```
VideoDigitizerError VDSetTimeBase (  
    VideoDigitizerComponent    ci,  
    TimeBase                    t );
```

ci

Identifies the application’s connection to the video digitizer component. An application obtains this value from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

t

A time base identifier. You can get this value from `NewTimeBase` (II–1119).

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Compressed Source Devices” (V–2846)

VDSetupBuffers

Defines output buffers for use with asynchronous grabs.

```
VideoDigitizerError VDSetupBuffers (
    VideoDigitizerComponent    ci,
    VdigBufferRecListHandle    bufferList );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

bufferList

A handle to a `VdigBufferRecList` (IV–2499) structure. Video digitizer components extract information about the spatial characteristics of the video destinations from these buffers.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

If you are developing a video digitizer component, note that the `matrix` field in the buffer list structure contains a pointer to the `MatrixRecord` (IV–2304) structure. It is your responsibility to copy that matrix structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Digitization” (V–2848)

See Also

Applications free these buffers by calling `VDReleaseAsyncBuffers` (III–2029).

VDSetWhiteLevelValue

Sets the white level value.

VDUseSafeBuffers

```
VideoDigitizerError VDSaveWhiteLevelValue (
    VideoDigitizerComponent ci,
    unsigned short          *whiteLevel );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

whiteLevel

A pointer to an integer that contains the new white level value. White level values range from 0 to 65,535, where 0 represents the minimum white value and 65,535 represents the maximum white value.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `QuickTimeComponents.h`

Programming summary: “Controlling Analog Video” (V–2850)

See Also

For the corresponding get function, see `VDGetWhiteLevelValue` (III–2024).

VDUseSafeBuffers

Instructs a video digitizer to use protected buffers.

```
VideoDigitizerError VDUseSafeBuffers (
    VideoDigitizerComponent ci,
    Boolean                 useSafeBuffers );
```

ci

Specifies the video digitizer component for this operation. Applications can obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

useSafeBuffers

Pass `TRUE` to use protected buffers; pass `FALSE` otherwise.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

VDUseThisCLUT

Specifies the lookup table for color digitization.

```
VideoDigitizerError VDUseThisCLUT (
    VideoDigitizerComponent    ci,
    CTabHandle                  colorTableHandle );
```

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

colorTableHandle

A handle to a `ColorTable` (IV–2210) structure. The video digitizer component uses the color table referred to by this parameter.

function result See “Error Codes” (IV–2663). Returns `noErr` if there is no error.

Discussion

This feature is useful only for capturing 8-bit color video.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QuickTimeComponents.h

Programming summary: “Controlling Color” (V–2849)

VideoMediaGetCodecParameter

Undocumented

```
ComponentResult VideoMediaGetCodecParameter (
    MediaHandler    mh,
    CodecType       cType,
    OSType          parameterID,
    Handle          outParameterData );
```

mh

A reference to a video media handler. You obtain this reference from `GetMediaHandler` (I–432).

VideoMediaGetStallCount

cType

A valid codec type constant; see “Codec Identifiers” (IV–2655).

parameterID

Undocumented

outParameterData

A handle to the returned data.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: `Movies.h`

See Also

For the corresponding set function, see `VideoMediaSetCodecParameter` (III–2060).

VideoMediaGetStallCount

Undocumented

```
ComponentResult VideoMediaGetStallCount (
    MediaHandler      mh,
    unsigned long      *stalls );
```

mh

A reference to a video media handler. You obtain this reference from `GetMediaHandler` (I–432).

stalls

The number of stalls.

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

VideoMediaGetStatistics

Returns the play-back frame rate of a movie.

```
ComponentResult VideoMediaGetStatistics (
    MediaHandler    mh );
```

mh

A reference to a video media handler. You obtain this reference from `GetMediaHandler` (I-432).

function result The average frame rate since the last time `VideoMediaResetStatistics` (III-2059) was called. Because of sampling errors, the values returned from this function are accurate only after waiting at least one second after calling `VideoMediaResetStatistics`.

Discussion

This function can only be used on video or MPEG media handlers. Because not all QuickTime movies have a constant frame rate, the results of this call can be difficult to interpret correctly. For this reason, the results of this function should not be displayed in a place where a novice user is likely to see it.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing the Video Frame Playback Rate” (V-2745)

Related Java Methods

`quicktime.std.movies.media.VideoMediaHandler.getStatistics()`

VideoMediaResetStatistics

Resets the video media handler’s counters before using `VideoMediaGetStatistics` (III-2059) to determine the frame rate of a movie.

```
ComponentResult VideoMediaResetStatistics (
    MediaHandler    mh );
```

mh

A reference to a video media handler. You obtain this reference from `GetMediaHandler` (I-432).

VideoMediaSetCodecParameter

function result You can access Movie Toolbox error returns through `GetMoviesError` (I–492) and `GetMoviesStickyError` (I–494), as well as in the function result. See “Error Codes” (IV–2663).

Special Considerations

This call can only be used on video or MPEG media handlers.

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: `Movies.h`

Programming summary: “Managing the Video Frame Playback Rate” (V–2745)

Related Java Methods

`quicktime.std.movies.media.VideoMediaHandler.resetStatistics()`

VideoMediaSetCodecParameter

Undocumented

```
ComponentResult VideoMediaSetCodecParameter (
    MediaHandler      mh,
    CodecType         cType,
    OSType             parameterID,
    long               parameterChangeSeed,
    void               *dataPtr,
    long               dataSize );
```

mh

A reference to a video media handler. You obtain this reference from `GetMediaHandler` (I–432).

cType

A valid codec type constant; see “Codec Identifiers” (IV–2655).

parameterID

Undocumented

parameterChangeSeed

Undocumented

dataPtr

A pointer to the data to be set.

dataSize

The size of the data.

function result You can access Movie Toolbox error returns through GetMoviesError (I-492) and GetMoviesStickyError (I-494), as well as in the function result. See “Error Codes” (IV-2663).

Version Notes

Introduced in QuickTime 4.

Programming Info

C interface file: Movies.h

See Also

For the corresponding get function, see VideoMediaGetCodecParameter (III-2057).

W

WinEventToMacEvent

Converts a Windows event message into a Macintosh event record.

```
long WinEventToMacEvent (
    void          *winMsg,
    EventRecord    *macEvent );
```

winMsg

A pointer to a Windows event message.

macEvent

A pointer to an EventRecord (IV-2246) structure.

function result *Undocumented*

Discussion

The following sample code shows how to convert Windows messages to Macintosh events, then pass those events to the QuickTime movie controller:

```
// WinEventToMacEvent coding example
// See “Discovering QuickTime,” page 240
MovieController mc; // Movie controller for movie

LRESULT CALLBACK WndProc
    (HWND hwnd, // Handle to window
    UINT iMsg, // Message type
```

XMLParseAddAttribute

```
        WPARAM      wParam,          // Message-dependent parameter
        LPARAM      lParam)         // Message-dependent parameter
{
    MSG             msg;              // Windows message structure
    EventRecord     er;               // Macintosh event record
    DWORD           dwPos;            // Mouse coordinates of message
    msg.hwnd        = hwnd;           // Window handle
    msg.message      = iMsg;           // Message type
    msg.wParam       = wParam;        // Word-length parameter
    msg.lParam       = lParam;        // Long-word parameter

    msg.time = GetMessageTime();       // Get time of message
    dwPos = GetMessagePos();           // Get mouse position
    msg.pt.x = LOWORD(dwPos);          // Extract x coordinate
    msg.pt.y = HIWORD(dwPos);          // Extract y coordinate

    WinEventToMacEvent(&msg, &er);    // Convert to event
    MCIsPlayerEvent(mc, &er);         // Pass event to QuickTime

    switch (iMsg) {                    // Dispatch on message type

        . . .                          // Handle message according to type

    } // end switch (iMsg)

} // end WndProc
```

Version Notes

Introduced in QuickTime 3 or earlier.

Programming Info

C interface file: QTML.h

X

XMLParseAddAttribute

Undocumented

```

ComponentResult XMLParseAddAttribute (
    ComponentInstance    aParser,
    UInt32               elementID,
    UInt32               nameSpaceID,
    char                 *attributeName,
    UInt32               *attributeID );

```

aParser

Undocumented

elementID

Undocumented

nameSpaceID

Undocumented

attributeName

Undocumented

attributeID

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseAddAttributeAndValue

Undocumented

```

ComponentResult XMLParseAddAttributeAndValue (
    ComponentInstance    aParser,
    UInt32               elementID,
    UInt32               nameSpaceID,
    char                 *attributeName,
    UInt32               *attributeID,
    UInt32               attributeValueKind,
    void                 *attributeValueKindInfo );

```

aParser

Undocumented

XMLParseAddAttributeValueKind

elementID

Undocumented

nameSpaceID

Undocumented

attributeName

Undocumented

attributeID

Undocumented

attributeValueKind

Undocumented

attributeValueKindInfo

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseAddAttributeValueKind

Undocumented

```
ComponentResult XMLParseAddAttributeValueKind (
    ComponentInstance  aParser,
    UInt32             elementID,
    UInt32             attributeID,
    UInt32             attributeValueKind,
    void               *attributeValueKindInfo );
```

aParser

Undocumented

elementID

Undocumented

attributeID

Undocumented

attributeValueKind

Undocumented

attributeValueKindInfo

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseAddElement

Undocumented

```
ComponentResult XMLParseAddElement (
    ComponentInstance  aParser,
    char               *elementName,
    UInt32             nameSpaceID,
    UInt32             *elementID,
    long               elementFlags );
```

aParser

Undocumented

elementName

Undocumented

nameSpaceID

Undocumented

elementID

Undocumented

elementFlags

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseAddMultipleAttributes

XMLParseAddMultipleAttributes

Undocumented

```
ComponentResult XMLParseAddMultipleAttributes (  
    ComponentInstance    aParser,  
    UInt32               elementID,  
    UInt32               *nameSpaceIDs,  
    char                 *attributeNames,  
    UInt32               *attributeIDs );
```

aParser

Undocumented

elementID

Undocumented

nameSpaceIDs

Undocumented

attributeNames

Undocumented

attributeIDs

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseAddMultipleAttributesAndValues

Undocumented

```
ComponentResult XMLParseAddMultipleAttributesAndValues (  
    ComponentInstance    aParser,  
    UInt32               elementID,  
    UInt32               *nameSpaceIDs,  
    char                 *attributeNames,  
    UInt32               *attributeIDs,  
    UInt32               *attributeValueKinds,  
    void                 **attributeValueKindInfos );
```


aParser

Undocumented

elementID

Undocumented

nameSpaceIDs

Undocumented

attributeNames

Undocumented

attributeIDs

Undocumented

attributeValueKinds

Undocumented

attributeValueKindInfos

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseAddNameSpace

Undocumented

```
ComponentResult XMLParseAddNameSpace (
    ComponentInstance  aParser,
    char               *nameSpaceURL,
    UInt32              *nameSpaceID );
```

aParser

Undocumented

nameSpaceURL

Undocumented

nameSpaceID

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

XMLParseDataRef

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseDataRef

Undocumented

```
ComponentResult XMLParseDataRef (  
    ComponentInstance  aParser,  
    Handle             dataRef,  
    OSType             dataRefType,  
    long               parseFlags,  
    XMLDoc             *document );
```

aParser

Undocumented

dataRef

Undocumented

dataRefType

Undocumented

parseFlags

Undocumented

document

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseDisposeXMLDoc

Undocumented

```
ComponentResult XMLParseDisposeXMLDoc (  
    ComponentInstance  aParser,
```

```
XMLDoc          document );
```

aParser

Undocumented

document

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseFile

Undocumented

```
ComponentResult XMLParseFile (
    ComponentInstance  aParser,
    ConstFSSpecPtr     fileSpec,
    long               parseFlags,
    XMLDoc             *document );
```

aParser

Undocumented

fileSpec

Undocumented

parseFlags

Undocumented

document

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseGetDetailedParseError

XMLParseGetDetailedParseError

Undocumented

```
ComponentResult XMLParseGetDetailedParseError (
    ComponentInstance  aParser,
    long               *errorLine,
    StringPtr          errDesc );
```

aParser

Undocumented

errorLine

Undocumented

errDesc

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseSetCharDataHandler

Undocumented

```
ComponentResult XMLParseSetCharDataHandler (
    ComponentInstance  aParser,
    CharDataHandlerUPP charData );
```

aParser

Undocumented

charData

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseSetCommentHandler

Undocumented

```
ComponentResult XMLParseSetCommentHandler (
    ComponentInstance    aParser,
    CommentHandlerUPP    comment );
```

aParser

Undocumented

comment

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseSetEndDocumentHandler

Undocumented

```
ComponentResult XMLParseSetEndDocumentHandler (
    ComponentInstance    aParser,
    EndDocumentHandlerUPP endDocument );
```

aParser

Undocumented

endDocument

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h



XMLParseSetEndElementHandler

XMLParseSetEndElementHandler

Undocumented

```
ComponentResult XMLParseSetEndElementHandler (
    ComponentInstance    aParser,
    EndElementHandlerUPP endElement );
```

aParser

Undocumented

endElement

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseSetEventParseRefCon

Undocumented

```
ComponentResult XMLParseSetEventParseRefCon (
    ComponentInstance    aParser,
    long                 refcon );
```

aParser

Undocumented

refcon

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseSetOffsetAndLimit

Undocumented

```
ComponentResult XMLParseSetOffsetAndLimit (
    ComponentInstance    aParser,
    UInt32               offset,
    UInt32               limit );
```

aParser

Undocumented

offset

Undocumented

limit

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseSetPreprocessInstructionHandler

Undocumented

```
ComponentResult XMLParseSetPreprocessInstructionHandler (
    ComponentInstance    aParser,
    PreprocessInstructionHandlerUPP preprocessInstruction );
```

aParser

Undocumented

preprocessInstruction

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseSetStartDocumentHandler

XMLParseSetStartDocumentHandler

Undocumented

```
ComponentResult XMLParseSetStartDocumentHandler (
    ComponentInstance      aParser,
    StartDocumentHandlerUPP startDocument );
```

aParser

Undocumented

startDocument

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

XMLParseSetStartElementHandler

Undocumented

```
ComponentResult XMLParseSetStartElementHandler (
    ComponentInstance      aParser,
    StartElementHandlerUPP startElement );
```

aParser

Undocumented

startElement

Undocumented

function result See “Error Codes” (IV–2663). Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

Callbacks

ActionsProc

Action callback for a media handler.

```
typedef OSErr (*ActionsProcPtr) (void *refcon, Track targetTrack,
long targetRefCon, QTEventRecordPtr theEvent);
```

```
// Declaration of a typical application-defined function
OSErr MyActionsProc (
    void          *refcon
    Track          targetTrack
    long           targetRefCon
    QTEventRecordPtr theEvent );
```

refcon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

targetTrack

The track in which to perform the actions.

targetRefCon

A reference constant for the target track.

theEvent

Pointer to a QTEventRecord (IV–2353) structure.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the MediaSetActionsCallback (II–888) and NewActionsUPP (II–1053) functions.

ActivateYDProc

Controls the highlighting of dialog items that are defined by your application and that can receive keyboard input.

```
typedef void (*ActivateYDProcPtr) (DialogPtr theDialog, short itemNo,
```

ActivateYDProc

```
Boolean activating, void *yourDataPtr);

// Declaration of a typical application-defined function
void MyActivateYDProc (
    DialogPtr    theDialog
    short        itemNo
    Boolean      activating
    void         *yourDataPtr );
```

theDialog

A pointer to a dialog box.

itemNo

An item number in the dialog box.

activating

A Boolean value that specifies whether the field is being activated (TRUE) or deactivated (FALSE).

yourDataPtr

A pointer to your own data.

Discussion

The two standard keyboard-input fields are the filename field (present only in Save dialog boxes) and the display list. Unless you override it through your own dialog hook function, the **Standard File Package** handles the highlighting of its own items and TextEdit fields. When the user changes the keyboard target by pressing the mouse button or the Tab key, the Standard File Package calls your `ActivateYDProc` callback twice: the first call specifies which field is being deactivated, and the second specifies which field is being activated.

Your application is responsible for removing the highlighting when one of its fields becomes inactive and for adding the highlighting when one of its fields becomes active. The Standard File Package can handle the highlighting of all TextEdit fields, even those defined by your application. You can also use your `ActivateYDProc` callback to keep track of which field is receiving keyboard input, if your application needs that information.

Special Considerations

Ordinarily, you need to supply an `ActivateYDProc` callback only if your application builds a list from which the user can select entries. The **Standard File Package** supplies the activation procedure for the file display list and for all TextEdit fields. You can also use your `ActivateYDProc` callback to keep track of which field is receiving keyboard input, if your application needs that information.

See Also

See the `CustomGetFilePreview` (I-163) function. For information about dialog boxes and the **Standard File Package**, see *Inside Macintosh: Files* (listed in the **bibliography**).

ComponentMPWorkFunctionProc

Undocumented

```
typedef ComponentResult (*ComponentMPWorkFunctionProcPtr)
(void *globalRefCon, ComponentMPWorkFunctionHeaderRecordPtr header);
```

```
// Declaration of a typical application-defined function
ComponentResult MyComponentMPWorkFunctionProc (
    void *globalRefCon
    ComponentMPWorkFunctionHeaderRecordPtr header );
```

globalRefCon

Undocumented

header

Pointer to a `ComponentMPWorkFunctionHeaderRecord` (IV-2215) structure.

function result See “Error Codes” (IV-2663). Your callback should return `noErr` if there is no error.

See Also

See the `ImageCodecGetBaseMPWorkFunction` (II-709) and `NewComponentMPWorkFunctionUPP` (II-1056) functions.

ComponentRoutineProc

Undocumented

```
typedef ComponentResult (*ComponentRoutineProcPtr) (ComponentParameters *cp,
Handle componentStorage);
```

```
// Declaration of a typical application-defined function
ComponentResult MyComponentRoutineProc (
    ComponentParameters *cp
    Handle componentStorage );
```

cp

Undocumented

DataHCompletionProc

componentStorage

Undocumented

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the RegisterComponent (III–1451) and NewComponentRoutineUPP (II–1057) functions.

DataHCompletionProc

Called upon completion of a read or write operation.

```
typedef void (*DataHCompletionProcPtr) (Ptr request, long refcon, OSErr err);
```

```
// Declaration of a typical application-defined function
```

```
void MyDataHCompletionProc (
```

```
    Ptr      request
```

```
    long     refcon
```

```
    OSErr    err );
```

request

Specifies a pointer to the data that was associated with the read request DataHScheduleData (I–220) or write request DataHWrite (I–232). The client program uses this pointer to determine which request has completed.

refcon

A reference constant that the client program supplied to your data handler component when it made the original request.

err

Indicates the success or failure of the operation. If the operation succeeded, set this parameter to 0. Otherwise, specify an appropriate error code.

Discussion

Data handler completion functions are guaranteed to be called at non-interrupt time. This means that you can safely call functions that are not interrupt-safe, such as DataHScheduleData (I–220), from within a completion function.

See Also

See the DataHGetFileSizeAsync (I–196), DataHReadAsync (I–217), DataHScheduleData (I–220), DataHWrite (I–232), and NewDataHCompletionUPP (II–1057) functions.

DlgHookProc

Handles item selections in a dialog box.

```
typedef short (*DlgHookProcPtr) (short item, DialogPtr theDialog);
```

// Declaration of a typical application-defined function

```
short MyDlgHookProc (  
    short      item  
    DialogPtr  theDialog );
```

item

The item number (see below).

theDialog

A pointer to the dialog record.

function result Set equal to the item number if your dialog hook function does not handle a selection.

item Constants

sfItemOpenButton

Save or Open button.

sfItemCancelButton

Cancel button.

sfItemBalloonHelp

Balloon Help.

sfItemVolumeUser

Volume icon and name.

sfItemEjectButton

Eject button.

sfItemDesktopButton

Desktop button.

sfItemFileListUser

Display list.

sfItemPopUpMenuUser

Directory pop-up menu.

sfItemDividerLinePict

Dividing line between buttons.

sfItemFileNameTextEdit

Filename field.

DlgHookYDProc

sfItemPromptStaticText
 Filename prompt text area.

sfItemNewFolderUser
 New Folder button.

Discussion

Your dialog hook function returns as its function result an integer that is either the item number passed to it or some other item number. If it returns one of the item numbers in the list of constants (see above), the **Standard File Package** handles the selected item as described in *Inside Macintosh: Files* (listed in the **bibliography**). If your dialog hook function does not handle a selection, it should pass the item number back to the Standard File Package for processing by setting its return value equal to the item number.

See Also

See the SFGetFilePreview (III–1674) and SFPGetFilePreview (III–1676) functions.

DlgHookYDProc

Handles user selections in a dialog box.

```
typedef short (*DlgHookYDProcPtr) (short item, DialogPtr theDialog,  
void *yourDataPtr);
```

```
// Declaration of a typical application-defined function  
short MyDlgHookYDProc (  
    short        item  
    DialogPtr    theDialog  
    void         *yourDataPtr );
```

item

 The dialog item number selected (see below).

theDialog

 A pointer to the dialog record.

yourDataPtr

 A pointer to the data received from your application, if any.

function result Set equal to the item number if your dialog hook function does not handle a selection.

item Constants

sfItemOpenButton
 Save or Open button.

<code>sfItemCancelButton</code>	Cancel button.
<code>sfItemBalloonHelp</code>	Balloon Help.
<code>sfItemVolumeUser</code>	Volume icon and name.
<code>sfItemEjectButton</code>	Eject button.
<code>sfItemDesktopButton</code>	Desktop button.
<code>sfItemFileListUser</code>	Display list.
<code>sfItemPopUpMenuUser</code>	Directory pop-up menu.
<code>sfItemDividerLinePict</code>	Dividing line between buttons.
<code>sfItemFileNameTextEdit</code>	Filename field.
<code>sfItemPromptStaticText</code>	Filename prompt text area.
<code>sfItemNewFolderUser</code>	New Folder button.

Discussion

Your dialog hook function returns as its function result an integer that is either the item number passed to it or some other item number. If it returns one of the item numbers in the list of constants (see above), the **Standard File Package** handles the selected item as described in *Inside Macintosh: Files* (listed in the **bibliography**). If your dialog hook function does not handle a selection, it should pass the item number back to the Standard File Package for processing by setting its return value equal to the item number.

See Also

See the `CustomGetFilePreview` (I-163) function.

DoMCActionProc

DoMCActionProc

Undocumented

```
typedef OSErr (*DoMCActionProcPtr) (void *refcon, short action, void *params, Boolean *handled);
```

```
// Declaration of a typical application-defined function
OSErr MyDoMCActionProc (
    void      *refcon
    short     action
    void      *params
    Boolean    *handled );
```

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

action

Undocumented

params

Undocumented

handled

A pointer to a Boolean in which you put TRUE if the action was handled, FALSE otherwise.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the `MCGetDoActionsProc` (II–811), `MediaSetDoMCActionCallback` (II–892), `MovieMediaGetDoMCActionCallback` (II–970), `MovieMediaGetChildDoMCActionCallback` (II–966), and `NewDoMCActionUPP` (II–1058) functions.

FileFilterProc

Determines which files the user can open.

```
typedef Boolean (*FileFilterProcPtr) (CInfoPBPtr pb);
```

```
// Declaration of a typical application-defined function
Boolean MyFileFilterProc (
    CInfoPBPtr    pb );
```


pb

A pointer to a catalog information parameter block. See Chapter 2 of *Inside Macintosh: Files* (listed in the **bibliography**).

function result TRUE suppresses display of the filename, and a value of FALSE allows the display.

Discussion

When QuickTime is displaying the contents of a volume or folder, it checks the file type of each file and filters out files whose types do not match your application's specifications. If your application also supplies a file filter function, the **Standard File Package** calls that function each time it identifies a file of an acceptable type. When your file filter function is called, it is passed, in the pb parameter, a pointer to a catalog information parameter block. See *Inside Macintosh: Files* (listed in the **bibliography**) for a description of the fields of this parameter block. Your function evaluates the catalog information parameter block and returns a Boolean value that determines whether the file is filtered (that is, a value of TRUE suppresses display of the filename, and a value of FALSE allows the display). If you do not supply a file filter function, the Standard File Package displays all files of the specified types.

See Also

See the StandardGetFilePreview (III–1910) and SFPGetFilePreview (III–1676) functions.

FileFilterYDProc

Determines which files the user can open, with a pointer to application-defined data.

```
typedef Boolean (*FileFilterYDProcPtr) (CInfoPBPtr pb, void *yourDataPtr);
```

```
// Declaration of a typical application-defined function
Boolean MyFileFilterYDProc (
    CInfoPBPtr    pb
    void          *yourDataPtr );
```

pb

A pointer to a catalog information parameter block. See Chapter 2 of *Inside Macintosh: Files* (listed in the **bibliography**).

yourDataPtr

A pointer to the data received from your application, if any.

function result TRUE suppresses display of the filename, and a value of FALSE allows

FilePlayCompletionProc

the display.

Discussion

When QuickTime is displaying the contents of a volume or folder, it checks the file type of each file and filters out files whose types do not match your application's specifications. If your application also supplies a file filter function, the **Standard File Package** calls that function each time it identifies a file of an acceptable type. When your file filter function is called, it is passed, in the pb parameter, a pointer to a catalog information parameter block. See *Inside Macintosh: Files* (listed in the **bibliography**) for a description of the fields of this parameter block. Your function evaluates the catalog information parameter block and returns a Boolean value that determines whether the file is filtered (that is, a value of TRUE suppresses display of the filename, and a value of FALSE allows the display). If you do not supply a file filter function, the Standard File Package displays all files of the specified types.

See Also

See the CustomGetFilePreview (I-163) function.

FilePlayCompletionProc

Obsolete.

```
typedef void (*FilePlayCompletionProcPtr) (SndChannelPtr chan);
```

```
// Declaration of a typical application-defined function
void MyFilePlayCompletionProc (
    SndChannelPtr    chan );
```

chan

A pointer to the sound channel.

GetMissingComponentResourceProc

Returns a specified component **resource**.

```
typedef OSErr (*GetMissingComponentResourceProcPtr) (Component c,
    OSType resType, short resID, void *refCon, Handle *resource);
```

```
// Declaration of a typical application-defined function
OSErr MyGetMissingComponentResourceProc (
    Component    c
    OSType       resType
    short        resID
```

```
void      *refCon
Handle    *resource );
```

c

A component identifier that specifies the component containing the **resource**. The caller obtains this identifier from `FindNextComponent` (I–360). If the caller registers a component, it can also obtain a component identifier from `RegisterComponent` (III–1451) or `RegisterComponentResource` (III–1453).

resType

A Mac OS **resource** type.

resID

The **resource** ID.

refCon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

resource

On return, a pointer to a handle to the **resource**.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

See Also

See the `GetComponentPublicResourceList` (I–393) and `NewGetMissingComponentResourceUPP` (II–1059) functions.

GetMovieProc

Provides movie data to the Movie Toolbox.

```
typedef OSErr (*GetMovieProcPtr) (long offset, long size, void *dataPtr,
void *refCon);
```

```
// Declaration of a typical application-defined function
OSErr MyGetMovieProc (
    long    offset
    long    size
    void    *dataPtr
    void    *refCon );
```

ICMAlignmentProc

offset

Specifies the offset into the movie **resource** (not the movie file). This is the location from which your function retrieves the movie data.

size

Specifies the amount of data requested by the toolbox, in bytes.

dataPtr

Specifies the destination for the movie data.

refCon

Contains a reference constant (defined as a void pointer). This is the same value you provided to the toolbox when you called `NewMovieFromUserProc` (II–1087).

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

Discussion

Normally, when a movie is loaded from a file (for example, by means of the `NewMovieFromFile` function), the toolbox uses that file as the default data reference. Since `NewMovieFromUserProc` (II–1087) does not require a file specification, your application should specify the file to be used as the default data reference using the `defaultDataRef` and `dataRefType` parameters.

Special Considerations

The toolbox automatically sets the movie’s graphics world based upon the current graphics port. Be sure that your application’s graphics world is valid before you call this function.

See Also

See the `NewMovieFromUserProc` (II–1087) and `NewGetMovieUPP` (II–1060) functions.

ICMAlignmentProc

Provides an alignment behavior for windows based on the screen’s bit depth.

```
typedef void (*ICMAlignmentProcPtr) (Rect *rp, long refcon);
```

```
// Declaration of a typical application-defined function
void MyICMAlignmentProc (
    Rect    *rp
    long    refcon );
```

rp

Contains a pointer to a rectangle that has already been aligned with a default alignment function.

refcon

Contains a reference constant value for use by your alignment function. Your application specifies the value of this reference constant in the alignment function structure you pass to the Image Compression Manager.

See Also

See the `ICMAlignmentProcRecord` (IV–2278) structure and the `AlignScreenRect` (I–46), `AlignWindow` (I–47), `DragAlignedGrayRgn` (I–304), `DragAlignedWindow` (I–306), and similar functions.

ICMCompletionProc

Called by a compressor component upon completion of an asynchronous operation.

```
typedef void (*ICMCompletionProcPtr) (OSErr result, short flags,
long refcon);
```

```
// Declaration of a typical application-defined function
void MyICMCompletionProc (
    OSErr    result
    short    flags
    long     refcon );
```

result

Indicator of success of current operation.

flags

Contains flags (see below) that indicate which part of the operation is complete. Note that more than one of the flags may be set to 1.

refcon

Contains a reference constant value for use by your completion function. Your application specifies the value of this reference constant in the callback function structure you pass to the Image Compression Manager.

flags Constants

`codecCompletionSource`

The Image Compression Manager is done with the source buffer. The Image Compression Manager sets this flag to 1 when it is done with the processing associated with the source buffer. For compression operations, the source is the

ICMConvertDataFormatProc

uncompressed pixel map you are compressing. For decompression operations, the source is the decompressed data you are decompressing.

codecCompletionDest

The Image Compression Manager is done with the destination buffer. The Image Compression Manager sets this flag to 1 when it is done with the processing associated with the destination buffer.

See Also

See the ICMCompletionProcRecord (IV-2279) structure and the CompressSequenceFrame (I-124), DecompressSequenceFrameS (I-243), DecompressSequenceFrameWhen (I-245), MediaQueueNonPrimarySourceData (II-885), and similar functions.

ICMConvertDataFormatProc

Undocumented

```
typedef OSErr (*ICMConvertDataFormatProcPtr) (void *refCon, long flags,  
Handle desiredFormat, Handle sourceDataFormat, void *srcData,  
long srcDataSize, void **dstData, long *dstDataSize);
```

```
// Declaration of a typical application-defined function  
OSErr MyICMConvertDataFormatProc (  
    void      *refCon  
    long      flags  
    Handle    desiredFormat  
    Handle    sourceDataFormat  
    void      *srcData  
    long      srcDataSize  
    void      **dstData  
    long      *dstDataSize );
```

refCon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

flags

Undocumented

desiredFormat

Undocumented

sourceDataFormat

Undocumented

srcData
Undocumented

srcDataSize
Undocumented

dstData
Undocumented

dstDataSize
Undocumented

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the CDSequenceNewDataSource (I–82) and NewICMConvertDataFormatUPP (II–1061) functions.

ICMCursorShieldedProc

Undocumented

```
typedef void (*ICMCursorShieldedProcPtr) (const Rect *r, void *refcon,
long flags);
```

```
// Declaration of a typical application-defined function
void MyICMCursorShieldedProc (
    const Rect    *r
    void          *refcon
    long          flags );
```

r
Undocumented

refcon
Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

flags
Undocumented

See Also

See the NewICMCursorShieldedUPP (II–1062) function.

ICMDataProc

Supplies compressed data during a decompression operation.

```
typedef OSErr (*ICMDataProcPtr) (Ptr *dataP, long bytesNeeded, long refcon);
```

```
// Declaration of a typical application-defined function
OSErr MyICMDataProc (
    Ptr      *dataP
    long     bytesNeeded
    long     refcon );
```

dataP

Contains a pointer to the address of the data buffer. The decompressor uses this parameter to indicate where your data-loading function should return the compressed data. You establish this data buffer when you start the decompression operation. For example, the *data* parameter to *FDecompressImage* (I-355) defines the location of the data buffer for that operation. Upon return from your data-loading function, this pointer should refer to the beginning of the compressed data that you loaded. The decompressor may also use this parameter to indicate that it wants to reset the mark within the compressed data stream. If the *dataP* parameter is set to *NIL*, the *bytesNeeded* parameter contains the new mark position, relative to the current position of the data stream. If your data-loading function does not support this operation, return a nonzero result code.

bytesNeeded

Specifies the number of bytes requested or the new mark offset. If the decompressor has requested additional compressed data (that is, the value of the *dataP* parameter is not *NIL*), then this parameter specifies how many bytes to return. This value never exceeds the size of the original data buffer. Your data-loading function should read the data from the current mark in the input data stream. If the decompressor has requested to set a new mark position in the data stream (that is, the value of the *dataP* parameter is *NIL*), then this parameter specifies the new mark position relative to the current position of the data stream.

refcon

Contains a reference constant value for use by your data-loading function. Your application specifies the value of this reference constant in the data-loading function structure you pass to the Image Compression Manager.

function result See “Error Codes” (IV-2663). Your callback should return *noErr* if there is no error.

See Also

See the ICMDataProcRecord (IV–2279) structure and the SetCompressedPixMapInfo (III–1573), SetDSequenceDataProc (III–1584), TrimImage (III–1956), ImageCodecGetCompressedImageSize (II–710), NewICMDataUPP (II–1062), and similar functions.

ICMFlushProc

Writes compressed data to a storage device during a compression operation.

```
typedef OSErr (*ICMFlushProcPtr) (Ptr data, long bytesAdded, long refcon);
```

```
// Declaration of a typical application-defined function
OSErr MyICMFlushProc (
    Ptr      data
    long     bytesAdded
    long     refcon );
```

data

Points to the data buffer. The compressor uses this parameter to indicate where your data-unloading function can find the compressed data. You establish this data buffer when you start the compression operation. For example, the data parameter to FCompressImage (I–344) defines the location of the data buffer for that operation. This pointer contains a **32-bit clean** address. Your ICMFlushProc function should make no other assumptions about the value of this address. The compressor may also use this parameter to indicate that it wants to reset the mark within the compressed data stream. If the data parameter is set to NIL, the bytesNeeded parameter contains the new mark position, relative to the current position of the output data stream. If your ICMFlushProc function does not support this operation, return a nonzero result code.

bytesAdded

Specifies the number of bytes to write or the new mark offset. If the compressor wants to write out some compressed data (that is, the value of data is not NIL), then this parameter specifies how many bytes to write. This value never exceeds the size of the original data buffer. Your ICMFlushProc function should write that data at the current mark in the output data stream. If the compressor has requested to set a new mark position in the output data stream (that is, the value of data is NIL), then this parameter specifies the new mark position relative to the current position of the data stream.

ICMMemoryDisposedProc

refcon

Contains a reference constant value for use by your `ICMFlushProc` function. Your application specifies the value of this reference constant in the data-unloading function structure you pass to the Image Compression Manager.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

Discussion

You assign an `ICMFlushProc` function to an image or a sequence by passing a pointer to a structure that identifies the function to the appropriate compression function.

See Also

See the `ICMFlushProcRecord` (IV–2280) structure and the `FCompressImage` (I–344), `ImageCodecTrimImage` (II–743), `SetCSequenceFlushProc` (III–1575), and similar functions.

ICMMemoryDisposedProc

Called before disposing of the memory allocated by a codec.

```
typedef void (*ICMMemoryDisposedProcPtr) (Ptr memoryBlock, void *refcon);
```

```
// Declaration of a typical application-defined function
void MyICMMemoryDisposedProc (
    Ptr      memoryBlock
    void     *refcon );
```

memoryBlock

Pointer to a block of memory.

refcon

Contains a reference constant value that your codec must pass to the `memoryGoneProc` function.

Discussion

This function must be called if the memory block is to be disposed of by the codec instead of by `ImageCodecDisposeMemory` (II–696). For example, this would occur if the codec is closed and still has memory allocation outstanding or if the memory is required to complete another operation.

Special Considerations

The Image Compression Manager does not currently track memory allocations. When a compressor or decompressor component instance is closed, it must ensure

that all blocks allocated by that instance are disposed and call your ICMMemoryDisposedProc (III–2092). This callback must not be called at interrupt time.

See Also

See the CDSequenceNewMemory (I–84), ImageCodecNewImageBufferMemory (II–727), ImageCodecNewMemory (II–729), and NewICMMemoryDisposedUPP (II–1063) functions.

ICMProgressProc

Reports on the progress of a compressor or decompressor.

```
typedef OSErr (*ICMProgressProcPtr) (short message, Fixed completeness,
long refcon);
```

```
// Declaration of a typical application-defined function
OSErr MyICMProgressProc (
    short    message
    Fixed    completeness
    long     refcon );
```

message

Indicates why the Image Compression Manager called your function. There are three valid messages, listed below.

completeness

Contains a fixed-point value indicating how far the operation has progressed. Its value is always between 0.0 and 1.0. This parameter is valid only when the message field is set to codecProgressUpdatePercent.

refcon

Contains a reference constant value for use by your **progress function**. Your application specifies the value of this reference constant in the progress function structure you pass to the Image Compression Manager.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error. When a component calls your **progress function**, it supplies you with a number that indicates the completion percentage. Your program can cause the component to terminate the current operation by returning a result code of codecAbortErr.

message Constants

codecProgressOpen

Indicates the start of a long operation. This is always the first message sent to your function. Your function can use this message to trigger the display of your progress window.

ImageCodecDrawBandCompleteProc

codecProgressUpdatePercent

Passes completion information to your function. The Image Compression Manager repeatedly sends this message to your function. The completeness parameter indicates the relative completion of the operation. You can use this value to update your progress window.

codecProgressClose

Indicates the end of a long operation. This is always the last message sent to your function. Your function can use this message as an indication to remove its progress window.

Discussion

The Image Compression Manager calls your **progress function** only during long operations, and it does not call your function more than 30 times per second.

See Also

The following functions have parameters that allow you to provide application-defined **progress functions**: FCompressImage (I–344), FDecompressImage (I–355), TrimImage (III–1956), FCompressPicture (I–348), FCompressPictureFile (I–352), DrawPictureFile (I–310), DrawTrimmedPicture (I–311), DrawTrimmedPictureFile (I–313), MakeThumbnailFromPicture (II–790), MakeThumbnailFromPictureFile (II–791), MakeThumbnailFromPixMap (II–792), SetCompressedPixMapInfo (III–1573), and GetCompressedPixMapInfo (I–397). If you pass a value of –1 in the progressProc parameter of any of these functions, you obtain a standard progress function.

ImageCodecDrawBandCompleteProc

Undocumented

```
typedef void (*ImageCodecDrawBandCompleteProcPtr) (void *refcon,  
ComponentResult drawBandResult, UInt32 drawBandCompleteFlags);
```

```
// Declaration of a typical application-defined function  
void MyImageCodecDrawBandCompleteProc (  
void *refcon  
ComponentResult drawBandResult  
UInt32 drawBandCompleteFlags );
```

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

drawBandResult

Undocumented

drawBandCompleteFlags

Undocumented

Version Notes

Introduced in QuickTime 5.

See Also

See the NewImageCodecDrawBandCompleteUPP (II–1064) function.

ImageCodecMPDrawBandProc

Undocumented

```
typedef ComponentResult (*ImageCodecMPDrawBandProcPtr) (void *refcon,
ImageSubCodecDecompressRecord *drp);
```

```
// Declaration of a typical application-defined function
ComponentResult MyImageCodecMPDrawBandProc (
    void *refcon,
    ImageSubCodecDecompressRecord *drp );
```

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

drp

Pointer to an ImageSubCodecDecompressRecord (IV–2289) structure.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the ImageCodecGetBaseMPWorkFunction (II–709) and NewImageCodecMPDrawBandUPP (II–1065) functions.

ImageCodecTimeTriggerProc

ImageCodecTimeTriggerProc

Undocumented

```
typedef void (*ImageCodecTimeTriggerProcPtr) (void *refcon);
```

```
// Declaration of a typical application-defined function
void MyImageCodecTimeTriggerProc (
    void      *refcon );
```

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

See Also

See the `ImageCodecScheduleFrame` (II–736) and `NewImageCodecTimeTriggerUPP` (II–1065) functions.

MCActionFilterProc

Responds to movie controller actions.

```
typedef Boolean (*MCActionFilterProcPtr) (MovieController mc, short *action,
void *params);
```

```
// Declaration of a typical application-defined function
Boolean MyMCActionFilterProc (
    MovieController    mc
    short              *action
    void                *params );
```

mc

Specifies the movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

action

A movie controller action. For a list of actions, see Chapter 2 of *Inside Macintosh: QuickTime* Components (listed in the **bibliography**).

params

A pointer to a structure, such as `QTStatusStringRecord` (IV–2387) or `ResolvedQTEventSpec` (IV–2401), that passes information to the callback. See `Movies.h`.

function result *Undocumented*

Discussion

Movie controller components allow your application to field movie controller actions. You define an `MCActionFilterProc` in your application and assign it to a controller by calling the `MCSetActionFilterWithRefCon` function.

See Also

See the `MCSetActionFilter` and `NewMCActionFilterUPP` (II–1068) functions.

MCActionFilterWithRefConProc

Responds to movie controller actions with a reference constant.

```
typedef Boolean (*MCActionFilterWithRefConProcPtr) (MovieController mc,
short action, void *params, long refCon);
```

```
// Declaration of a typical application-defined function
Boolean MyMCActionFilterWithRefConProc (
    MovieController    mc
    short              action
    void               *params
    long               refCon );
```

mc

Specifies the movie controller for the operation. You obtain this identifier from `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131), or from `NewMovieController` (II–1071).

action

A movie controller action. For a list of actions, see Chapter 2 of *Inside Macintosh: QuickTime Components* (listed in the **bibliography**).

params

A pointer to a structure, such as `QTStatusStringRecord` (IV–2387), that passes information to the callback. See `Movies.h`.

ModalFilterProc

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result *Undocumented*

See Also

See the `MCSetActionFilterWithRefCon` (II-829) and `NewMCActionFilterWithRefConUPP` (II-1069) functions.

ModalFilterProc

Determines how events are filtered for modal dialog boxes.

```
typedef Boolean (*ModalFilterProcPtr) (DialogPtr theDialog,  
EventRecord *theEvent, DialogItemIndex *itemHit);
```

```
// Declaration of a typical application-defined function  
Boolean MyModalFilterProc (  
    DialogPtr      theDialog  
    EventRecord    *theEvent  
    DialogItemIndex *itemHit );
```

theDialog

A pointer to the dialog record.

theEvent

A pointer to the event record.

itemHit

The item number.

function result Your `ModalFilterProc` callback returns a `Boolean` value that reports whether it handled the event. If your function returns a value of `FALSE`, `QuickTime` processes the event through its own filters. If your function returns a value of `TRUE`, `QuickTime` returns with no further action.

Discussion

The **Standard File Package** contains an internal filter function that performs some preliminary processing on each event it receives. If you provide a `ModalFilterProc` callback, it is called after the internal Standard File Package filter function and before the event is sent to your dialog hook function. You might provide a `ModalFilterProc` callback for several reasons. If you have customized the `Open` or

Save dialog boxes by adding one or more items, you might want to map some of the user's keypresses to those items in the same way that the internal filter function maps certain keypresses to existing items.

Special Considerations

You can supply a `ModalFilterProc` callback only when you use one of the procedures that displays a customized dialog box. Another reason to provide a `ModalFilterProc` callback is to avoid a problem that can arise if an update event is received for one of your application's windows while a **Standard File Package** dialog box is displayed.

See Also

See the `ImageCodecRequestSettings` (II–735), `QTVideoOutputCustomConfigureDisplay` (II–1322), `SFPGetFilePreview` (III–1676), and similar functions.

ModalFilterYDProc

Determines how the Dialog Manager filters events.

```
typedef Boolean (*ModalFilterYDProcPtr) (DialogPtr theDialog,
EventRecord *theEvent, short *itemHit, void *yourDataPtr);
```

```
// Declaration of a typical application-defined function
Boolean MyModalFilterYDProc (
    DialogPtr      theDialog
    EventRecord    *theEvent
    short          *itemHit
    void           *yourDataPtr );
```

`theDialog`

A pointer to the dialog record.

`theEvent`

A pointer to the event record.

`itemHit`

The item number.

`yourDataPtr`

A pointer to the data received from your application, if any.

function result Your `ModalFilterProc` callback returns a Boolean value that reports whether it handled the event. If your function returns a value of FALSE, QuickTime processes the event through its own filters. If your function returns a value of TRUE, QuickTime returns with no further

MovieDrawingCompleteProc

action.

Discussion

The `ModalFilterProc` callback used with custom file dialogs requires the additional `yourDataPtr` parameter.

See Also

See the `CustomGetFilePreview` (I–163) and `GraphicsExportRequestSettings` (I–588) functions.

MovieDrawingCompleteProc

Called when movie drawing is complete.

```
typedef OSErr (*MovieDrawingCompleteProcPtr) (Movie theMovie, long refCon);
```

```
// Declaration of a typical application-defined function
OSErr MyMovieDrawingCompleteProc (
    Movie    theMovie
    long     refCon );
```

`theMovie`

Specifies the movie for this operation.

`refCon`

Contains the reference constant you supplied when your application called the `SetMovieDrawingCompleteProc` function.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

Special Considerations

Some media handlers may take less efficient playback paths when a `MovieDrawingCompleteProc` is used, so it should be used only when absolutely necessary.

See Also

See the `SetMovieDrawingCompleteProc` (III–1617) and `NewMovieDrawingCompleteUPP` (II–1072) functions.

MovieExecuteWiredActionsProc

Undocumented

```
typedef OSErr (*MovieExecuteWiredActionsProcPtr) (Movie theMovie,
void *refcon, long flags, QTAtomContainer wiredActions);
```

```
// Declaration of a typical application-defined function
OSErr MyMovieExecuteWiredActionsProc (
    Movie          theMovie
    void          *refcon
    long          flags
    QTAtomContainer wiredActions );
```

theMovie

Specifies the movie for this operation.

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

flags

Undocumented

wiredActions

Undocumented

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the AddMovieExecuteWiredActionsProc (I–36), RemoveMovieExecuteWiredActionsProc (III–1457), and NewMovieExecuteWiredActionsUPP (II–1073) functions.

MovieExportGetDataProc

Defines a data source for an export operation.

```
typedef OSErr (*MovieExportGetDataProcPtr) (void *refCon,
MovieExportGetDataParams *params);
```

```
// Declaration of a typical application-defined function
OSErr MyMovieExportGetDataProc (
    void          *refCon
```

MovieExportGetPropertyProc

```
MovieExportGetDataParams    *params );
```

refCon

Contains the value passed to MovieExportAddDataSource (II–919) in the refCon parameter

params

The sample request is made through a MovieExportGetDataParams (IV–2314) structure.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

Discussion

This callback is passed to MovieExportAddDataSource (II–919) to define a new data source for an export operation. The function is used by the exporting application to request source media data to be used in the export operation. For example, in a video export operation, frames of video data (either compressed or uncompressed) are provided. In a sound export operation, buffers of audio (either compressed or uncompressed) are provided.

Special Considerations

The data pointed to by dataPtr must remain valid until the next call to this function. The MovieExportGetDataProc callback is responsible for allocating and disposing of the memory associated with this data pointer.

See Also

See the MovieExportAddDataSource (II–919), MovieExportNewGetDataAndPropertiesProcs (II–927), and MovieExportDisposeGetDataAndPropertiesProcs functions.

MovieExportGetPropertyProc

Returns parameters that determine the appropriate format for movie export data.

```
typedef OSErr (*MovieExportGetPropertyProcPtr) (void *refcon, long trackID,  
OSType propertyType, void *propertyValue);
```

```
// Declaration of a typical application-defined function  
OSErr MyMovieExportGetPropertyProc (  
    void        *refcon  
    long        trackID  
    OSType      propertyType  
    void        *propertyValue );
```

refcon

Contains the value passed to `MovieExportAddDataSource` (II–919) in the `refCon` parameter.

trackID

Specifies the value returned from `MovieExportAddDataSource` (II–919).

propertyType

Contains a pointer to the location of the requested property information.

propertyValue

Specifies the property being requested (see below).

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error. If this function doesn’t have a setting for a requested property, it should return an error.

propertyValue Constants

`scSoundSampleRateType`

An `UnsignedFixed` value that represents a sound track’s sampling rate.

`scSoundSampleSizeType`

A short integer that represents a sound track’s sample size.

`scSoundChannelCountType`

A short integer that represents a sound track’s channel count.

`scSoundCompressionType`

A sound track’s compression type constant; see “Codec Identifiers” (IV–2655).

`movieExportWidth`

A fixed integer that represents a video track’s image width in pixels.

`movieExportHeight`

A fixed integer that represents a video track’s image height in pixels.

`movieExportVideoFilter`

A pointer to a `QTAtomContainer` handle that references a video track’s filter atom container.

`scSpatialSettingsType`

A video track’s `SCSpatialSettings` (IV–2423) structure.

`scTemporalSettingsType`

A video track’s `SCTemporalSettings` (IV–2427) structure.

`scDataRateSettingsType`

A video track’s `SCDataRateSettings` (IV–2417) structure.

`movieExportDuration`

The `TimeRecord` (IV–2486) structure for the whole movie.

MoviePrePrerollCompleteProc

Discussion

This function is passed to `MovieExportAddDataSource` (II–919) to define a new data source for an export operation. For example, a video export operation may call this function to determine the dimensions of the destination video track. The export component provides a default value for the property based on the source data format. For example, if no values for video track width and height properties were provided by the callback function, the dimensions of the source data would be used.

See Also

See the `MovieExportAddDataSource` (II–919), `MovieExportNewGetDataAndPropertiesProcs` (II–927), `MovieExportDisposeGetDataAndPropertiesProcs` (II–920), and `NewMovieExportGetPropertyUPP` (II–1074) functions.

MoviePrePrerollCompleteProc

Undocumented

```
typedef void (*MoviePrePrerollCompleteProcPtr) (Movie theMovie,
OSErr prerollErr, void *refcon);
```

```
// Declaration of a typical application-defined function
void MyMoviePrePrerollCompleteProc (
    Movie    theMovie
    OSErr    prerollErr
    void     *refcon );
```

`theMovie`

Specifies the movie for this operation.

`prerollErr`

Undocumented

`refcon`

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

See Also

See the `PrePrerollMovie` (II–1141) and `NewMoviePrePrerollCompleteUPP` (II–1089) functions.

MoviePreviewCallOutProc

Controls the playing of a movie's **preview**.

```
typedef Boolean (*MoviePreviewCallOutProcPtr) (long refcon);
```

```
// Declaration of a typical application-defined function
Boolean MyMoviePreviewCallOutProc (
    long    refcon );
```

```
refcon
```

A reference constant you specified when you called `PlayMoviePreview` (II–1140).

function result If your function sets this value to `FALSE`, the Movie Toolbox stops the **preview** and returns to your application.

Discussion

If you call `GetMovieActiveSegment` (I–457) from within your callback, the Movie Toolbox can change the active movie segment to be the **preview** segment of the movie. The Movie Toolbox will restore the active segment when the preview is done playing.

See Also

See `PlayMoviePreview` (II–1140).

MovieProgressProc

Monitors the progress of the Movie Toolbox during long operations.

```
typedef OSErr (*MovieProgressProcPtr) (Movie theMovie, short message,
short whatOperation, Fixed percentDone, long refcon);
```

```
// Declaration of a typical application-defined function
OSErr MyMovieProgressProc (
    Movie    theMovie
    short    message
    short    whatOperation
    Fixed    percentDone
    long     refcon );
```

```
theMovie
```

Specifies the movie for this operation. The Movie Toolbox sets this parameter to identify the appropriate movie.

MovieProgressProc

message

Constant (see below) that indicates why the Movie Toolbox called your function.

whatOperation

Constant (see below) that indicates the long operation that is currently underway.

percentDone

Contains a fixed-point value indicating how far the operation has progressed. Its value is always between 0.0 and 1.0. This parameter is valid only when the message field is set to `movieProgressUpdatePercent`.

refcon

Reference constant value for use by your **progress function**. Your application specifies the value of this reference constant when you assign the progress function to the movie.

function result Your **progress function** should return an error value. The Movie Toolbox examines this value after each `movieProgressUpdatePercent` message and before continuing the current operation. Set this value to a nonzero value to cancel the operation; set it to `noErr` to continue.

message Constants

movieProgressOpen

Indicates the start of a long operation. This is always the first message sent to your function. Your function can use this message to trigger the display of your progress window.

movieProgressUpdatePercent

Passes completion information to your function. The Movie Toolbox repeatedly sends this message to your function. The `percentDone` parameter indicates the relative completion of the operation. You can use this value to update your progress window.

movieProgressClose

Indicates the end of a long operation. This is always the last message sent to your function. Your function can use this message as an indication to remove its progress window.

whatOperation Constants

progressOpFlatten

Your application has called the `FlattenMovie` (I-372) or `FlattenMovieData` (I-374) function.

progressOpInsertTrackSegment

Your application has called the InsertTrackSegment (II-764) function. The Movie Toolbox calls the **progress function** that is assigned to the movie that contains the destination track.

progressOpInsertMovieSegment

Your application has called the InsertMovieSegment (II-762) function. The Movie Toolbox calls the **progress function** that is assigned to the destination movie.

progressOpPaste

Your application has called the PasteMovieSelection (II-1139) function. The Movie Toolbox calls the **progress function** that is assigned to the destination movie.

progressOpAddMovieSelection

Your application has called the AddMovieSelection (I-38) function. The Movie Toolbox calls the **progress function** that is assigned to the destination movie. The Movie Toolbox calls the progress function that is assigned to the destination movie.

progressOpCopy

Your application has called the CopyMovieSelection (I-139) function. The Movie Toolbox calls the **progress function** that is assigned to the destination movie.

progressOpCut

Your application has called the CutMovieSelection (I-166) function. The Movie Toolbox calls the **progress function** that is assigned to the destination movie.

progressOpLoadMovieIntoRam

Your application has called the LoadMovieIntoRam (II-781) function. The Movie Toolbox calls the **progress function** that is assigned to the destination movie.

progressOpLoadTrackIntoRam

Your application has called the LoadTrackIntoRam (II-782) function. The Movie Toolbox calls the **progress function** that is assigned to the destination track.

progressOpLoadMediaIntoRam

Your application has called the LoadMediaIntoRam (II-779) function. The Movie Toolbox calls the **progress function** that is assigned to the destination media.

progressOpImportMovie

Your application has called the ConvertFileToMovieFile (I-129) function. The Movie Toolbox calls the **progress function** that is associated with the destination movie file. This flag is also used, as appropriate, for the PasteHandleIntoMovie (II-1137) functions.

MovieRgnCoverProc

progressOpExportMovie

Your application has called the `ConvertMovieToFile` (I–134) function. The Movie Toolbox calls the **progress function** that is associated with the destination movie. This flag is also used, as appropriate, for the `PutMovieIntoTypedHandle` (II–1152) function.

See Also

See the `ConvertFileToMovieFile` (I–129), `GetMovieProgressProc` (I–488), `MovieExportSetProgressProc` (II–930), `MovieImportSetProgressProc` (II–961), `SetMovieProgressProc` (III–1630), and `NewMovieProgressUPP` (II–1090) functions.

MovieRgnCoverProc

Undocumented

```
typedef OSErr (*MovieRgnCoverProcPtr) (Movie theMovie, RgnHandle changedRgn,
long refcon);
```

```
// Declaration of a typical application-defined function
OSErr MyMovieRgnCoverProc (
    Movie        theMovie
    RgnHandle    changedRgn
    long         refcon );
```

theMovie

Specifies the movie for this operation.

changedRgn

Undocumented

refcon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

See Also

See the `GetMovieCoverProcs` (I–464), `SetMovieCoverProcs` (III–1614), and `NewMovieRgnCoverUPP` (II–1091) functions.

MoviesErrorProc

An error-notification function, called each time the current error value is to be set to a nonzero value.

```
typedef void (*MoviesErrorProcPtr) (OSErr theErr, long refcon);
```

// Declaration of a typical application-defined function

```
void MyMoviesErrorProc (
    OSErr    theErr
    long     refcon );
```

theErr

An error code; see “Error Codes” (IV–2663).

refcon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

See Also

See the SetMoviesErrorProc (III–1633) and NewMoviesErrorUPP (II–1091) functions.

MusicMIDIReadHookProc

Undocumented

```
typedef ComponentResult (*MusicMIDIReadHookProcPtr) (MusicMIDIPacket *mp,
long myRefCon);
```

// Declaration of a typical application-defined function

```
ComponentResult MyMusicMIDIReadHookProc (
    MusicMIDIPacket *mp
    long             myRefCon );
```

mp

A MusicMIDIPacket (IV–2320) structure.

myRefCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

MusicMIDISendProc

See Also

See the `MusicReceiveMIDI` (II–1005), `NAUseDefaultMIDIInput` (II–1052), `QTMIDIUseReceivePort` (II–1189), and `NewMusicMIDIReadHookUPP` (II–1093) functions.

MusicMIDISendProc

Undocumented

```
typedef ComponentResult (*MusicMIDISendProcPtr) (ComponentInstance self,  
long refCon, MusicMIDIPacket *mmp);
```

```
// Declaration of a typical application-defined function  
ComponentResult MyMusicMIDISendProc (  
    ComponentInstance    self  
    long                  refCon  
    MusicMIDIPacket       *mmp );
```

self

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

mmp

A pointer to a `MusicMIDIPacket` (IV–2320) structure.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

See Also

See the `MusicGetMIDIProc` (II–998), `MusicDerivedSetMIDI` (II–978), `MusicSetMIDIProc` (II–1009), and `NewMusicMIDISendUPP` (II–1094) functions.

MusicOfflineDataProc

Undocumented

```
typedef ComponentResult (*MusicOfflineDataProcPtr) (Ptr SoundData,  
long numBytes, long myRefCon);
```

```
// Declaration of a typical application-defined function  
ComponentResult MyMusicOfflineDataProc (  
    Ptr SoundData  
    long numBytes  
    long myRefCon );
```

```
Ptr      SoundData
long     numBytes
long     myRefCon );
```

SoundData

Undocumented

numBytes

Undocumented

myRefCon

Undocumented

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the MusicStartOffline (II–1016) and NewMusicOfflineDataUPP (II–1094) functions.

PrePrerollCompleteProc

Undocumented

```
typedef void (*PrePrerollCompleteProcPtr) (MediaHandler mh, OSErr err,
void *refcon);
```

// Declaration of a typical application-defined function

```
void MyPrePrerollCompleteProc (
    MediaHandler    mh
    OSErr           err
    void            *refcon );
```

mh

A media handler.

err

An error code; see “Error Codes” (IV–2663).

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Proc

See Also

See the `MediaPrePrerollBegin` (II–882) and `NewMoviePrePrerollCompleteUPP` (II–1089) functions.

Proc

Generic callback.

```
typedef long (*ProcPtr) ();

// Declaration of a typical application-defined function
long MyProc ();
```

function result Undocumented

See Also

See the `NewComponentFunctionUPP` (II–1056) function.

QDArcProc

An application-defined low-level routine to draw arcs or wedges.

```
typedef void (*QDArcProcPtr) (GrafVerb verb, Rect *r, short startAngle,
short arcAngle);
```

```
// Declaration of a typical application-defined function
void MyQDArcProc (
    GrafVerb    verb
    Rect        *r
    short       startAngle
    short       arcAngle );
```

`verb`

A constant (see below) that defines the drawing action.

`r`

The rectangle to contain the arc.

`startAngle`

The beginning angle.

`arcAngle`

The ending angle.

verb Constants

kQDGrafVerbFrame
 Make a pen drawing.

kQDGrafVerbPaint
 Paint in the shape

kQDGrafVerbErase
 Use the background color.

kQDGrafVerbInvert
 Invert the drawing color.

kQDGrafVerbFill
 Use the fill pattern.

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDBitsProc

An application-defined low-level routine to do bit and pixel image transfers.

```
typedef void (*QDBitsProcPtr) (BitMap *srcBits, Rect *srcRect, Rect *dstRect,
short mode, RgnHandle maskRgn);
```

```
// Declaration of a typical application-defined function
void MyQDBitsProc (
    BitMap      *srcBits
    Rect        *srcRect
    Rect        *dstRect
    short       mode
    RgnHandle   maskRgn );
```

srcBits

A pointer to a bitmap or pixel map containing the image to copy.

srcRect

A pointer to the source rectangle.

dstRect

A pointer to the destination rectangle.

mode

The source mode for transferring; see “Graphics Transfer Modes” (IV–2670).

maskRgn

A handle to a region acting as a mask for the transfer.

QDCommentProc

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDCommentProc

An application-defined low-level routine to process picture comments, such as printer instructions.

```
typedef void (*QDCommentProcPtr) (short kind, short dataSize,  
Handle dataHandle);
```

```
// Declaration of a typical application-defined function  
void MyQDCommentProc (  
    short    kind  
    short    dataSize  
    Handle   dataHandle );
```

kind

A comment type code, usually specific to a printer or other device. For a discussion of picture comments, see Appendix B of *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

dataSize

The size of any additional data passed in the dataHandle parameter.

dataHandle

A handle to additional data, or NIL if there is none.

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDGetPicProc

An application-defined low-level routine to retrieve picture definitions.

```
typedef void (*QDGetPicProcPtr) (Ptr dataPtr, short byteCount);
```

```
// Declaration of a typical application-defined function  
void MyQDGetPicProc (  
    Ptr      dataPtr  
    short    byteCount );
```

dataPtr

A pointer to the picture data.

byteCount

The size of the picture data in bytes.

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDJShieldCursorProc

An application-defined low-level routine to hide the cursor.

```
typedef void (*QDJShieldCursorProcPtr) (short left, short top, short right,
short bottom);
```

```
// Declaration of a typical application-defined function
void MyQDJShieldCursorProc (
    short    left
    short    top
    short    right
    short    bottom );
```

left

The x coordinate of the upper-left corner of the cursor rectangle.

top

The y coordinate of the upper-left corner of the cursor rectangle.

right

The x coordinate of the lower-right corner of the cursor rectangle.

bottom

The y coordinate of the lower-right corner of the cursor rectangle.

QDLineProc

An application-defined low-level routine to draw a line.

```
typedef void (*QDLineProcPtr) (Point newPt);

// Declaration of a typical application-defined function
void MyQDLineProc (
    Point    newPt );
```

newPt

The point to which to draw the line.

QDOpcodeProc

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDOpcodeProc

An application-defined low-level routine to process QuickDraw opcodes.

```
typedef void (*QDOpcodeProcPtr) (Rect *fromRect, Rect *toRect, short opcode,  
short version);
```

```
// Declaration of a typical application-defined function  
void MyQDOpcodeProc (  
    Rect      *fromRect  
    Rect      *toRect  
    short     opcode  
    short     version );
```

fromRect

A pointer to the source rectangle.

toRect

A pointer to the destination rectangle.

opcode

The QuickDraw opcode. See Appendix A of *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

version

The format version. See Appendix A of *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

See Also

See the CQDProcs (IV–2226) structure.

QDOvalProc

An application-defined low-level routine to draw ovals.

```
typedef void (*QDOvalProcPtr) (GrafVerb verb, Rect *r);
```

```
// Declaration of a typical application-defined function  
void MyQDOvalProc (  
    GrafVerb  verb  
    Rect      *r );
```

verb

A constant (see below) that defines the drawing action.

r

The rectangle to contain the oval.

verb Constants

kQDGrafVerbFrame

Make a pen drawing.

kQDGrafVerbPaint

Paint in the shape

kQDGrafVerbErase

Use the background color.

kQDGrafVerbInvert

Invert the drawing color.

kQDGrafVerbFill

Use the fill pattern.

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDPixProc

Undocumented

```
typedef void (*QDPixProcPtr) (PixMap *src, Rect *srcRect,
MatrixRecord *matrix, short mode, RgnHandle mask, PixMap *matte,
Rect *matteRect, short flags);
```

// Declaration of a typical application-defined function

```
void MyQDPixProc (
    PixMap      *src
    Rect        *srcRect
    MatrixRecord *matrix
    short       mode
    RgnHandle   mask
    PixMap      *matte
    Rect        *matteRect
    short       flags );
```

src

Undocumented

QDPolyProc

srcRect
Undocumented

matrix
Undocumented

mode
Undocumented

mask
Undocumented

matte
Undocumented

matteRect
Undocumented

flags
Undocumented

See Also

Undocumented

QDPolyProc

An application-defined low-level routine to draw polygons.

```
typedef void (*QDPolyProcPtr) (GrafVerb verb, PolyHandle poly);
```

```
// Declaration of a typical application-defined function
void MyQDPolyProc (
    GrafVerb    verb
    PolyHandle   poly );
```

verb

A constant (see below) that defines the drawing action.

poly

A handle to the polygon data.

verb Constants

kQDGrafVerbFrame

Make a pen drawing.

kQDGrafVerbPaint

Paint in the shape

kQDGrafVerbErase

Use the background color.

kQDGrafVerbInvert

Invert the drawing color.

kQDGrafVerbFill

Use the fill pattern.

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDPutPicProc

An application-defined low-level routine to save information as a picture definition.

```
typedef void (*QDPutPicProcPtr) (Ptr dataPtr, short byteCount);
```

// Declaration of a typical application-defined function

```
void MyQDPutPicProc (
    Ptr      dataPtr
    short    byteCount );
```

dataPtr

A pointer to the data to be saved.

byteCount

The size of the data in bytes.

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDRectProc

An application-defined low-level routine to draw rectangles.

```
typedef void (*QDRectProcPtr) (GrafVerb verb, Rect *r);
```

// Declaration of a typical application-defined function

```
void MyQDRectProc (
    GrafVerb  verb
    Rect      *r );
```

QDRgnProc

verb

A constant (see below) that defines the drawing action.

r

A pointer to a Rect (IV–2399) structure that defines the rectangle.

verb Constants

kQDGrafVerbFrame

Make a pen drawing.

kQDGrafVerbPaint

Paint in the shape

kQDGrafVerbErase

Use the background color.

kQDGrafVerbInvert

Invert the drawing color.

kQDGrafVerbFill

Use the fill pattern.

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDRgnProc

An application-defined low-level routine to draw regions.

```
typedef void (*QDRgnProcPtr) (GrafVerb verb, RgnHandle rgn);
```

```
// Declaration of a typical application-defined function
```

```
void MyQDRgnProc (  
    GrafVerb    verb  
    RgnHandle   rgn );
```

verb

A constant (see below) that defines the drawing action.

rgn

A handle to a MacRegion (IV–2303) structure that defines the region.

verb Constants

kQDGrafVerbFrame

Make a pen drawing.

kQDGrafVerbPaint

Paint in the shape

kQDGrafVerbErase

Use the background color.

kQDGrafVerbInvert

Invert the drawing color.

kQDGrafVerbFill

Use the fill pattern.

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDRRectProc

An application-defined low-level routine to draw rounded rectangles.

```
typedef void (*QDRRectProcPtr) (GrafVerb verb, Rect *r, short ovalWidth,
short ovalHeight);
```

// Declaration of a typical application-defined function

```
void MyQDRRectProc (
    GrafVerb    verb
    Rect        *r
    short       ovalWidth
    short       ovalHeight );
```

verb

A constant (see below) that defines the drawing action.

r

A pointer to a Rect (IV–2399) structure that defines the rectangle.

ovalWidth

The width diameter for the corner oval.

ovalHeight

The height diameter for the corner oval.

verb Constants

kQDGrafVerbFrame

Make a pen drawing.

kQDGrafVerbPaint

Paint in the shape

kQDGrafVerbErase

Use the background color.

QDStdGlyphsProc

kQDGrafVerbInvert

Invert the drawing color.

kQDGrafVerbFill

Use the fill pattern.

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDStdGlyphsProc

An application-defined low-level routine to draw **Unicode** text.

```
typedef OSStatus (*QDStdGlyphsProcPtr) (void *dataStream, ByteCount size);
```

// Declaration of a typical application-defined function

```
OSStatus MyQDStdGlyphsProc (  
    void      *dataStream  
    ByteCount  size );
```

dataStream

A pointer to the **Unicode** data.

size

The size of the data in bytes.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the CQDProcs (IV–2226) structure.

QDTextProc

An application-defined low-level routine to draw text.

```
typedef void (*QDTextProcPtr) (short byteCount, Ptr textBuf, Point numer,  
    Point denom);
```

// Declaration of a typical application-defined function

```
void MyQDTextProc (  
    short      byteCount  
    Ptr        textBuf  
    Point      numer  
    Point      denom );
```


byteCount

The number of bytes of text to draw.

textBuf

A pointer to the buffer containing the text.

numer

The scaling numerator.

denom

The scaling denominator.

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QDTxMeasProc

An application-defined low-level routine to measure text width.

```
typedef short (*QDTxMeasProcPtr) (short byteCount, Ptr textAddr,
Point *numer, Point *denom, FontInfo *info);
```

// Declaration of a typical application-defined function

```
short MyQDTxMeasProc (
    short      byteCount
    Ptr        textAddr
    Point      *numer
    Point      *denom
    FontInfo   *info );
```

byteCount

The number of bytes of text data to measure.

textAddr

A pointer to the location of the text.

numer

The scaling numerator.

denom

The scaling denominator.

info

A pointer to a FontInfo (IV–2261) structure that provides measurement information for the current font.

function result The width of the text in pixels.

QTBandwidthNotificationProc

See Also

See the QDProcs (IV–2342) and CQDProcs (IV–2226) structures.

QTBandwidthNotificationProc

Undocumented

```
typedef OSErr (*QTBandwidthNotificationProcPtr) (long flags, void *reserved,  
void *refcon);
```

```
// Declaration of a typical application-defined function  
OSErr MyQTBandwidthNotificationProc (  
    long    flags  
    void    *reserved  
    void    *refcon );
```

flags

Undocumented

reserved

Reserved.

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the QTBandwidthRequest (II–1156), QTBandwidthRequestForTimeBase (II–1157), QTScheduledBandwidthRequest (II–1219), and NewQTBandwidthNotificationUPP (II–1096) functions.

QTCallbackProc

A generic callback function, installed by CallMeWhen (I–67).

```
typedef void (*QTCallbackProcPtr) (QTCallback cb, long refCon);
```

```
// Declaration of a typical application-defined function  
void MyQTCallbackProc (  
    QTCallback    cb
```

long refCon);

cb

A pointer to a CallBackRecord (IV–2165) structure containing the callback’s data.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

See Also

See the CallMeWhen (I–67) and NewQTCallBackUPP (II–1097) functions.

QTSMoDalFilterProc

Undocumented

```
typedef Boolean (*QTSMoDalFilterProcPtr) (DialogPtr inDialog,
const EventRecord *inEvent, SInt16 *ioItemHit, void *inRefCon);
```

```
// Declaration of a typical application-defined function
Boolean MyQTSMoDalFilterProc (
    DialogPtr      inDialog
    const EventRecord *inEvent
    SInt16         *ioItemHit
    void           *inRefCon );
```

inDialog

A pointer to the dialog record.

inEvent

A pointer to the event record.

ioItemHit

Undocumented

inRefCon

Undocumented

function result *Undocumented*

Version Notes

Introduced in QuickTime 5.

See Also

See the NewQTSMoDalFilterUPP (II–1097) function.

QTSNotificationProc

QTSNotificationProc

A back channel from a presentation to its creator, sending notification of various events such as a presentation, ending, or acknowledgment of a preroll request.

```
typedef ComponentResult (*QTSNotificationProcPtr) (ComponentResult inErr,  
OSType inNotificationType, void *inNotificationParams, void *inRefCon);
```

```
// Declaration of a typical application-defined function  
ComponentResult MyQTSNotificationProc (  
    ComponentResult    inErr  
    OSType              inNotificationType  
    void                *inNotificationParams  
    void                *inRefCon );
```

inErr

Undocumented

inNotificationType

The kind of notification; see QuickTimeStreaming.h.

inNotificationParams

Undocumented

inRefCon

Undocumented

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the QTSNewPresentationParams (IV–2376) and QTSMediaNotificationParams (IV–2374) structures and the QTSPresGetNotificationProc (II–1275), QTSPresSetNotificationProc (II–1296), and NewQTSNotificationUPP (II–1098) functions.

QTSPanelFilterProc

Undocumented

```
typedef Boolean (*QTSPanelFilterProcPtr) (QTSPanelFilterParams *inParams,  
void *inRefCon);
```

```
// Declaration of a typical application-defined function  
Boolean MyQTSPanelFilterProc (  
    QTSPanelFilterParams    *inParams
```

```
void                                *inRefCon );

inParams
    Undocumented

inRefCon
    Undocumented

function result    Undocumented
```

Version Notes

Introduced in QuickTime 5.

See Also

See the NewQTSPanelFilterUPP (II-1098) function.

QTSyncTaskProc

Undocumented

```
typedef void (*QTSyncTaskProcPtr) (void *task);

// Declaration of a typical application-defined function
void MyQTSyncTaskProc (
    void    *task );

task
    Undocumented
```

See Also

See the QTSyncTaskRecord (IV-2391) structure and the NewQTSyncTaskUPP (II-1099) function.

QTVRBackBufferImagingProc

An imaging procedure that draws directly into the back buffer for a QTVR panoramic node.

```
typedef OSErr (*QTVRBackBufferImagingProcPtr) (QTVRInstance qtvr,
Rect *drawRect, UInt16 areaIndex, UInt32 flagsIn, UInt32 *flagsOut,
SInt32 refCon);

// Declaration of a typical application-defined function
OSErr MyQTVRBackBufferImagingProc (
    QTVRInstance    qtvr
```

QTVREnteringNodeProc

Rect	*drawRect
UInt16	areaIndex
UInt32	flagsIn
UInt32	*flagsOut
SInt32	refCon);

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

drawRect

Undocumented

areaIndex

Undocumented

flagsIn

Undocumented

flagsOut

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the QTVRSetBackBufferImagingProc (II–1411) and NewQTVRBackBufferImagingUPP (II–1099) functions.

QTVREnteringNodeProc

A routine called whenever a QTVR node is entered, in response either to user actions or QuickTime VR Manager functions that change nodes.

```
typedef OSErr (*QTVREnteringNodeProcPtr) (QTVRInstance qtvr, UInt32 nodeID, SInt32 refCon);
```

```
// Declaration of a typical application-defined function
OSErr MyQTVREnteringNodeProc (
    QTVRInstance    qtvr
    UInt32          nodeID
    SInt32          refCon );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

nodeID

A node ID. Set this parameter to `kQTVRCurrentNode` to designate the current node.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

See Also

See the `QTVRSetEnteringNodeProc` (II–1419) and `NewQTVREnteringNodeUPP` (II–1100) functions.

QTVRImagingCompleteProc

An imaging completion procedure for a QuickTime VR movie, called whenever QTVR finishes drawing an image into the prescreen buffer.

```
typedef OSErr (*QTVRImagingCompleteProcPtr) (QTVRInstance qtvr,
SInt32 refCon);
```

```
// Declaration of a typical application-defined function
OSErr MyQTVRImagingCompleteProc (
    QTVRInstance    qtvr
    SInt32           refCon );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

QTVRInterceptProc

See Also

See the `QTVRSetPrescreenImagingCompleteProc` (II–1432) and `NewQTVRImagingCompleteUPP` (II–1100) functions.

QTVRInterceptProc

A routine that is called when certain QTVR functions are intercepted.

```
typedef void (*QTVRInterceptProcPtr) (QTVRInstance qtvr,  
QTVRInterceptPtr qtvrMsg, SInt32 refCon, Boolean *cancel);
```

```
// Declaration of a typical application-defined function  
void MyQTVRInterceptProc (  
    QTVRInstance    qtvr  
    QTVRInterceptPtr qtvrMsg  
    SInt32           refCon  
    Boolean          *cancel );
```

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (II–1373).

`qtvrMsg`

A pointer to a `QTVRInterceptRecord` (IV–2396) structure that determines which functions are intercepted.

`refCon`

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

`cancel`

A pointer to a Boolean; set to TRUE if the intercept is cancelled.

See Also

See the `QTVRInstallInterceptProc` (II–1387) and `NewQTVRInterceptUPP` (II–1101) functions.

QTVRLeavingNodeProc

A routine called whenever a QTVR node is left, in response either to user actions or QuickTime VR Manager functions that change nodes.

```
typedef OSErr (*QTVRLeavingNodeProcPtr) (QTVRInstance qtvr,
```



```
UInt32 fromNodeID, UInt32 toNodeID, Boolean *cancel, SInt32 refCon);
```

```
// Declaration of a typical application-defined function
```

```
OSErr MyQTVRLeavingNodeProc (
    QTVRInstance qtvr
    UInt32        fromNodeID
    UInt32        toNodeID
    Boolean        *cancel
    SInt32        refCon );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

fromNodeID

The ID of the node being left. Set this parameter to kQTVRCurrentNode to designate the current node.

toNodeID

The ID of the node being entered. Set this parameter to kQTVRCurrentNode to designate the current node.

cancel

A pointer to a Boolean; set to TRUE if the callback is cancelled.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the QTVRSetLeavingNodeProc (II–1428) and NewQTVRLeavingNodeUPP (II–1101) functions.

QTVRMouseOverHotSpotProc

A routine that is called when the mouse is over a hot spot in a QTVR movie.

```
typedef OSErr (*QTVRMouseOverHotSpotProcPtr) (QTVRInstance qtvr,
    UInt32 hotSpotID, UInt32 flags, SInt32 refCon);
```

```
// Declaration of a typical application-defined function
```

```
OSErr MyQTVRMouseOverHotSpotProc (
```

RTPMPDataReleaseProc

```
QTVRInstance    qtvr
UInt32          hotSpotID
UInt32          flags
SInt32          refCon );
```

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (II–1373).

hotSpotID

A hot spot ID.

flags

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the QTVRSetMouseOverHotSpotProc (II–1429) and NewQTVRMouseOverHotSpotUPP (II–1102) functions.

RTPMPDataReleaseProc

Routine called when a media packetizer is finished with its sample data.

```
typedef void (*RTPMPDataReleaseProcPtr) (UInt8 *inData, void *inRefCon);
```

```
// Declaration of a typical application-defined function
void MyRTPMPDataReleaseProc (
    UInt8    *inData
    void     *inRefCon );
```

inData

A pointer to the data.

inRefCon

A pointer to information passed from a RTPMPSampleDataParams (IV–2407) structure.

See Also

See the RTPMPSampleDataParams (IV–2407) structure and the NewRTPMPDataReleaseUPP (II–1102) function.

RTPPBCallbackProc

Routine used to communicate with the caller of a media packetizer.

```
typedef void (*RTPPBCallbackProcPtr) (OSType inSelector, void *ioParams,
void *inRefCon);
```

```
// Declaration of a typical application-defined function
void MyRTPPBCallbackProc (
    OSType    inSelector
    void      *ioParams
    void      *inRefCon );
```

inSelector
Undocumented

ioParams
Undocumented

inRefCon
Undocumented

See Also

See the RTPPBGetCallback (III–1493), RTPPBSetCallback (III–1497), and NewRTPPBCallbackUPP (II–1103) functions.

SCModalFilterProc

Filter routine called when a user event occurs in a sequence compression modal dialog box.

```
typedef Boolean (*SCModalFilterProcPtr) (DialogPtr theDialog,
EventRecord *theEvent, short *itemHit, long refcon);
```

```
// Declaration of a typical application-defined function
Boolean MySCModalFilterProc (
    DialogPtr    theDialog
    EventRecord  *theEvent
    short        *itemHit
    long         refcon );
```

SCModalHookProc

`theDialog`

A pointer to a dialog box.

`theEvent`

A pointer to an EventRecord (IV–2246) structure that defines a user event.

`itemHit`

A pointer to an item ID number in the dialog box.

`refcon`

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result Return TRUE if the event was handled, FALSE otherwise.

See Also

See the SCExtendedProcs (IV–2418) structure and the SCGetCompressionExtended and NewSCModalFilterUPP (II–1103) functions.

SCModalHookProc

Called whenever the user selects an item in the dialog box.

```
typedef short (*SCModalHookProcPtr) (DialogPtr theDialog, short itemHit,  
void *params, long refcon);
```

// Declaration of a typical application-defined function

```
short MySCModalHookProc (  
    DialogPtr    theDialog  
    short        itemHit  
    void         *params  
    long         refcon );
```

`theDialog`

A pointer to a dialog box.

`itemHit`

A pointer to an item ID number in the dialog box.

`params`

A pointer to your data area.

`refcon`

A reference constant that the client code supplies to your callback.

function result Return TRUE if the event was handled, FALSE otherwise.

Discussion

You can use this callback to customize the operation of the standard image-compression dialog box. For example, you might want to support a custom button that activates a secondary dialog box. Another possibility would be to provide additional validation support when the user clicks OK.

See Also

See the `SCExtendedProcs` (IV–2418) structure and the `SCGetCompressionExtended` (III–1544) and `NewSCModalHookUPP` (II–1104) functions.

SequenceFilterDataProc

Undocumented

```
typedef ComponentResult (*SequenceFilterDataProcPtr) (Ptr data, long len,
long sampleCount, TimeValue timeStamp, void *sampleExtra, void *refCon);
```

// Declaration of a typical application-defined function

```
ComponentResult MySequenceFilterDataProc (
```

```
    Ptr          data
    long         len
    long         sampleCount
    TimeValue    timeStamp
    void         *sampleExtra
    void         *refCon );
```

`data`

Undocumented

`len`

Undocumented

`sampleCount`

Undocumented

`timeStamp`

Undocumented

`sampleExtra`

Undocumented

`refCon`

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if

SFModalFilterProc

there is no error.

SFModalFilterProc

Undocumented

```
typedef Boolean (*SFModalFilterProcPtr) (DialogPtr theDialog,  
EventRecord *theEvent, short *itemHit, long refcon);
```

```
// Declaration of a typical application-defined function  
Boolean MySFModalFilterProc (  
    DialogPtr    theDialog  
    EventRecord  *theEvent  
    short        *itemHit  
    long         refcon );
```

theDialog

A pointer to a dialog box.

theEvent

Undocumented

itemHit

Undocumented

refcon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result Return TRUE if the event was handled, FALSE otherwise.

SGAddFrameBottleProc

Undocumented

```
typedef ComponentResult (*SGAddFrameBottleProcPtr) (SGChannel c,  
short bufferNum, TimeValue atTime, TimeScale scale,  
const SGCompressInfo *ci, long refCon);
```

```
// Declaration of a typical application-defined function  
ComponentResult MySGAddFrameBottleProc (  
    SGChannel    c  
    short        bufferNum  
    TimeValue    atTime
```

```

TimeScale      scale
const SGCompressInfo *ci
long           refCon );

```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

bufferNum

A buffer identifier provided by the sequence grabber component.

atTime

Undocumented

scale

The current time scale.

ci

A pointer to a `SGCompressInfo` (IV–2432) structure.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

See Also

See the `VideoBottles` (IV–2501) structure and the `NewSGAddFrameBottleUPP` (II–1104) function.

SGCompressBottleProc

Undocumented

```

typedef ComponentResult (*SGCompressBottleProcPtr) (SGChannel c,
short bufferNum, long refCon);

```

```

// Declaration of a typical application-defined function
ComponentResult MySGCompressBottleProc (
    SGChannel    c
    short        bufferNum
    long         refCon );

```

SGCompressCompleteBottleProc

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

bufferNum

A buffer identifier provided by the sequence grabber component.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

See Also

See the `VideoBottles` (IV–2501) structure and the `SGCompressBottleUPP` (`api>SGCompressBottleUPP–SGCompressBottleUPP`) function.

SGCompressCompleteBottleProc

Undocumented

```
typedef ComponentResult (*SGCompressCompleteBottleProcPtr) (SGChannel c,  
short bufferNum, Boolean *done, SGCompressInfo *ci, long refCon);
```

```
// Declaration of a typical application-defined function  
ComponentResult MySGCompressCompleteBottleProc (  
    SGChannel      c  
    short          bufferNum  
    Boolean        *done  
    SGCompressInfo *ci  
    long           refCon );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

bufferNum

A buffer identifier provided by the sequence grabber component.

done

A pointer to a Boolean; return `TRUE` if the task was completed, `FALSE` otherwise.

ci

A pointer to a `SGCompressInfo` (IV–2432) structure.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the VideoBottles (IV–2501) structure and the SGCompressCompleteBottleUPP (api>SGCompressCompleteBottleUPP–SGCompressCompleteBottleUPP) function.

SGDataProc

Undocumented

```
typedef OSErr (*SGDataProcPtr) (SGChannel c, Ptr p, long len, long *offset,
long chRefCon, TimeValue time, short writeType, long refCon);
```

// Declaration of a typical application-defined function

```
OSErr MySGDataProc (
    SGChannel    c
    Ptr          p
    long         len
    long         *offset
    long         chRefCon
    TimeValue    time
    short        writeType
    long         refCon );
```

c

The connection identifier for the channel for this operation. You get this value from SGNewChannel (III–1753) or SGNewChannelFromComponent (III–1756).

p

Undocumented

len

Undocumented

offset

Undocumented

chRefCon

Undocumented

SGDisplayBottleProc

time

Undocumented

writeType

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the SGSetDataProc (III–1787) and NewSGDataUPP (II–1106) functions.

SGDisplayBottleProc

Undocumented

```
typedef ComponentResult (*SGDisplayBottleProcPtr) (SGChannel c,  
short bufferNum, MatrixRecord *mp, RgnHandle clipRgn, long refCon);
```

// Declaration of a typical application-defined function

```
ComponentResult MySGDisplayBottleProc (
```

```
    SGChannel      c  
    short          bufferNum  
    MatrixRecord   *mp  
    RgnHandle      clipRgn  
    long           refCon );
```

c

The connection identifier for the channel for this operation. You get this value from SGNewChannel (III–1753) or SGNewChannelFromComponent (III–1756).

bufferNum

A buffer identifier provided by the sequence grabber component.

mp

Undocumented

clipRgn

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the VideoBottles (IV–2501) structure and the SGDisplayBottleUPP (api>SGDisplayBottleUPP–SGDisplayBottleUPP) function.

SGDisplayCompressBottleProc

Undocumented

```
typedef ComponentResult (*SGDisplayCompressBottleProcPtr) (SGChannel c,
Ptr dataPtr, ImageDescriptionHandle desc, MatrixRecord *mp,
RgnHandle clipRgn, long refCon);
```

// Declaration of a typical application-defined function
ComponentResult MySGDisplayCompressBottleProc (

```
    SGChannel      c
    Ptr            dataPtr
    ImageDescriptionHandle desc
    MatrixRecord   *mp
    RgnHandle      clipRgn
    long           refCon );
```

c

The connection identifier for the channel for this operation. You get this value from SGNewChannel (III–1753) or SGNewChannelFromComponent (III–1756).

dataPtr

Undocumented

desc

Undocumented

mp

Undocumented

clipRgn

Undocumented

SGGrabBottleProc

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the VideoBottles (IV–2501) structure and the SGDisplayCompressBottleUPP (api>SGDisplayCompressBottleUPP–SGDisplayCompressBottleUPP) function.

SGGrabBottleProc

Undocumented

```
typedef ComponentResult (*SGGrabBottleProcPtr) (SGChannel c,  
short bufferNum, long refCon);
```

```
// Declaration of a typical application-defined function  
ComponentResult MySGGrabBottleProc (  
    SGChannel    c  
    short        bufferNum  
    long         refCon );
```

c

The connection identifier for the channel for this operation. You get this value from SGNewChannel (III–1753) or SGNewChannelFromComponent (III–1756).

bufferNum

A buffer identifier provided by the sequence grabber component.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the VideoBottles (IV–2501) structure and the SGGrabBottleUPP (api>SGGrabBottleUPP–SGGrabBottleUPP) function.

SGGrabCompleteBottleProc

Undocumented

```
typedef ComponentResult (*SGGrabCompleteBottleProcPtr) (SGChannel c,
short bufferNum, Boolean *done, long refCon);
```

```
// Declaration of a typical application-defined function
ComponentResult MySGGrabCompleteBottleProc (
    SGChannel    c
    short        bufferNum
    Boolean      *done
    long         refCon );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

bufferNum

A buffer identifier provided by the sequence grabber component.

done

A pointer to a Boolean; return TRUE if the task was completed, FALSE otherwise.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

See Also

See the `VideoBottles` (IV–2501) structure and the `SGGrabCompleteBottleUPP` (`api>SGGrabCompleteBottleUPP–SGGrabCompleteBottleUPP`) function.

SGGrabCompressCompleteBottleProc

Undocumented

```
typedef ComponentResult (*SGGrabCompressCompleteBottleProcPtr) (SGChannel c,
Boolean *done, SGCompressInfo *ci, TimeRecord *t, long refCon);
```

```
// Declaration of a typical application-defined function
ComponentResult MySGGrabCompressCompleteBottleProc (
    SGChannel    c
```

SGModalFilterProc

```
Boolean      *done
SGCompressInfo *ci
TimeRecord   *t
long         refCon );
```

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (III–1753) or `SGNewChannelFromComponent` (III–1756).

done

A pointer to a Boolean; return TRUE if the task was completed, FALSE otherwise.

ci

A pointer to a `SGCompressInfo` (IV–2432) structure.

t

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

See Also

See the `VideoBottles` (IV–2501) structure and the `SGGrabCompressCompleteBottleUPP` (`api>SGGrabCompressCompleteBottleUPP–SGGrabCompressCompleteBottleUPP`) function.

SGModalFilterProc

Undocumented

```
typedef Boolean (*SGModalFilterProcPtr) (DialogPtr theDialog,
const EventRecord *theEvent, short *itemHit, long refCon);
```

```
// Declaration of a typical application-defined function
```

```
Boolean MySGModalFilterProc (
    DialogPtr      theDialog
    const EventRecord *theEvent
    short          *itemHit
    long           refCon );
```

theDialog

A pointer to a dialog box.

theEvent

Undocumented

itemHit

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result Return TRUE if the event was handled, FALSE otherwise.

See Also

See the SGSettingsDialog (III–1808), SGPanelSetEventFilter (III–1767), and NewSGModalFilterUPP (II–1109) functions.

SGTransferFrameBottleProc

Undocumented

```
typedef ComponentResult (*SGTransferFrameBottleProcPtr) (SGChannel c,
short bufferNum, MatrixRecord *mp, RgnHandle clipRgn, long refCon);
```

// Declaration of a typical application-defined function

```
ComponentResult MySGTransferFrameBottleProc (
```

```
    SGChannel      c
    short          bufferNum
    MatrixRecord   *mp
    RgnHandle      clipRgn
    long           refCon );
```

c

The connection identifier for the channel for this operation. You get this value from SGNewChannel (III–1753) or SGNewChannelFromComponent (III–1756).

bufferNum

A buffer identifier provided by the sequence grabber component.

mp

Undocumented

clipRgn

Undocumented

SICCompletionProc

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

See Also

See the `VideoBottles` (IV–2501) structure and the `SGTransferFrameBottleUPP` (`api>SGTransferFrameBottleUPP–SGTransferFrameBottleUPP`) function.

SICCompletionProc

Accesses the parameter block for a read request from the Sound Manager to a custom sound input component.

```
typedef void (*SICCompletionProcPtr) (SndInputCmpParamPtr SICParmPtr);
```

```
// Declaration of a typical application-defined function
void MySICCompletionProc (
    SndInputCmpParamPtr    SICParmPtr );
```

SICParmPtr

A pointer to a `SndInputCmpParam` (IV–2446) structure.

Discussion

Use this callback only if you are writing your own sound input component.

See Also

See the `SndInputCmpParam` (IV–2446) structure.

SICompletionProc

A completion callback called by the sound input functions.

```
typedef void (*SICompletionProcPtr) (SPBPtr inParamPtr);
```

```
// Declaration of a typical application-defined function
void MySICompletionProc (
    SPBPtr    inParamPtr );
```


inParamPtr

A pointer to an SPB (IV–2456) structure.

See Also

See the SPB (IV–2456) structure and the NewSICompletionUPP (II–1110) function.

SIInterruptProc

An interrupt callback called by the sound input functions.

```
typedef void (*SIInterruptProcPtr) (SPBPtr inParamPtr, Ptr dataBuffer,
short peakAmplitude, long sampleSize);
```

// Declaration of a typical application-defined function

```
void MySIInterruptProc (
    SPBPtr    inParamPtr
    Ptr       dataBuffer
    short     peakAmplitude
    long      sampleSize );
```

inParamPtr

A pointer to an SPB (IV–2456) structure.

dataBuffer

A pointer to data.

peakAmplitude

Undocumented

sampleSize

Undocumented

See Also

See the SPB (IV–2456) structure and the NewSIInterruptUPP (II–1110) function.

SndCallbackProc

The callback that the Sound Manager executes whenever it receives a callBackCmd command.

```
typedef void (*SndCallbackProcPtr) (SndChannelPtr chan, SndCommand *cmd);
```

// Declaration of a typical application-defined function

```
void MySndCallbackProc (
    SndChannelPtr    chan
```

SndDoubleBackProc

```
SndCommand      *cmd );
```

chan

A pointer to a SndChannel (IV–2440) structure.

cmd

A pointer to a SndCommand (IV–2441) structure.

See Also

See the SndChannel (IV–2440) structure and the SndNewChannel (III–1839) and NewSndCallBackUPP (II–1111) functions. The callBackCmd command is listed in “Sound Commands” (IV–2691).

SndDoubleBackProc

Undocumented

```
typedef void (*SndDoubleBackProcPtr) (SndChannelPtr channel,  
SndDoubleBufferPtr doubleBufferPtr);
```

```
// Declaration of a typical application-defined function  
void MySndDoubleBackProc (  
    SndChannelPtr      channel  
    SndDoubleBufferPtr doubleBufferPtr );
```

channel

A pointer to a SndChannel (IV–2440) structure.

doubleBufferPtr

A pointer to a SndDoubleBuffer (IV–2442) structure.

See Also

See the SndDoubleBufferHeader (IV–2443) and SndDoubleBufferHeader2 (IV–2444) structures and the NewSndDoubleBackUPP (II–1111) function.

SoundConverterFillBufferDataProc

Provides data to the Sound Converter.

```
typedef Boolean (*SoundConverterFillBufferDataProcPtr)  
(SoundComponentDataPtr *data, void *refCon);
```

```
// Declaration of a typical application-defined function  
Boolean MySoundConverterFillBufferDataProc (  

```

```
SoundComponentDataPtr    *data
void                     *refCon );
```

data

A pointer to a SoundComponentData (IV–2448) structure.

refCon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result *Undocumented*

See Also

See the SoundConverterFillBuffer (III–1864) and NewSoundConverterFillBufferDataUPP (II–1112) functions.

SoundParamProc

A callback that is called by a sound output device when another buffer of audio data is needed or when the sound has finished playing.

```
typedef Boolean (*SoundParamProcPtr) (SoundParamBlockPtr *pb);
```

```
// Declaration of a typical application-defined function
Boolean MySoundParamProc (
    SoundParamBlockPtr    *pb );
```

pb

A pointer to a SoundParamBlock (IV–2454) structure.

function result *Undocumented*

See Also

See the SoundParamBlock (IV–2454) structure and the NewSoundParamUPP (II–1112) function.

StdPixProc

Undocumented

```
typedef void (*StdPixProcPtr) (PixMap *src, Rect *srcRect,
MatrixRecord *matrix, short mode, RgnHandle mask, PixMap *matte,
Rect *matteRect, short flags);
```

TextMediaProc

```
// Declaration of a typical application-defined function
void MyStdPixProc (
    PixMap      *src
    Rect        *srcRect
    MatrixRecord *matrix
    short       mode
    RgnHandle    mask
    PixMap      *matte
    Rect        *matteRect
    short       flags );
```

src

Undocumented

srcRect

Undocumented

matrix

Undocumented

mode

Undocumented

mask

Undocumented

matte

Undocumented

matteRect

Undocumented

flags

Undocumented

See Also

See the NewStdPixUPP (II-1118) function.

TextMediaProc

A callback that can be called whenever a text sample is displayed in a movie.

```
typedef OSErr (*TextMediaProcPtr) (Handle theText, Movie theMovie,
    short *displayFlag, long refcon);
```

```
// Declaration of a typical application-defined function
OSErr MyTextMediaProc (
```

```

Handle    theText
Movie     theMovie
short     *displayFlag
long      refcon );

```

theText

A handle to the text being displayed.

theMovie

Specifies the movie for this operation.

displayFlag

Undocumented

refcon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if there is no error.

See Also

See the `TextMediaSetTextProc` (III–1947) and `NewTextMediaUPP` (II–1118) functions.

TrackTransferProc

Called when a track is forced to draw into a particular graphics world, which may be different from that of the movie.

```
typedef OSErr (*TrackTransferProcPtr) (Track t, long refCon);
```

// Declaration of a typical application-defined function

```

OSErr MyTrackTransferProc (
    Track    t
    long     refCon );

```

t

A track designator.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

function result See “Error Codes” (IV–2663). Your callback should return `noErr` if

TuneCallbackProc

there is no error.

See Also

See the `SetTrackGWorld` (III–1659) and `NewTrackTransferUPP` (II–1122) functions.

TuneCallbackProc

Called when a sequence of music events is placed into a queue to be played.

```
typedef void (*TuneCallbackProcPtr) (const TuneStatus *status, long refCon);
```

```
// Declaration of a typical application-defined function
void MyTuneCallbackProc (
    const TuneStatus *status
    long refCon );
```

status

A pointer to a `TuneStatus` (IV–2492) structure.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

See Also

See the `TuneQueue` (III–1964) and `NewTuneCallbackUPP` (II–1122) functions.

TunePlayCallbackProc

Supports the `TuneSetNoteChannels` (III–1969) function.

```
typedef void (*TunePlayCallbackProcPtr) (unsigned long *event, long seed,
long refCon);
```

```
// Declaration of a typical application-defined function
void MyTunePlayCallbackProc (
    unsigned long *event
    long seed
    long refCon );
```

event

A pointer to a QuickTime music event structure in the sequence data.

seed

A 32-bit value that is guaranteed to be different for each call to the callback routine (unless 2³² calls are made, after which the values repeat), with one exception: the value passed at the beginning of a note is also passed at the end of the note's duration, together with a note structure or an extended note in which the velocity bits are set to 0.

refCon

A reference constant that the client code supplies to the callback.

See Also

See the `TuneSetNoteChannels` (III–1969) and `NewTunePlayCallBackUPP` (II–1123) functions.

TweenerDataProc

A callback the **tween** component calls with the value generated by a tween operation.

```
typedef ComponentResult (*TweenerDataProcPtr) (TweenRecord *tr,
void *tweenData, long tweenDataSize, long dataDescriptionSeed,
Handle dataDescription, ICMCompletionProcRecordPtr asyncCompletionProc,
UniversalProcPtr transferProc, void *refCon);
```

// Declaration of a typical application-defined function

```
ComponentResult MyTweenerDataProc (
    TweenRecord          *tr
    void                 *tweenData
    long                 tweenDataSize
    long                 dataDescriptionSeed
    Handle                dataDescription
    ICMCompletionProcRecordPtr asyncCompletionProc
    UniversalProcPtr      transferProc
    void                 *refCon );
```

tr

A pointer to the **tween** record for the tween operation.

tweenData

A pointer to the generated **tween** value.

tweenDataSize

The size, in bytes, of the **tween** value.

VdigIntProc

dataDescriptionSeed

The starting value for the calculation. Every time the content of the dataDescription handle changes, this value should be incremented.

dataDescription

Specifies a handle containing a description of the **tween** value passed. For basic types such as integers, the calling tween component should set this parameter to NIL. For more complex types such as compressed image data, the calling tween component should set this handle to contain a description of the tween value, such as an image description.

asyncCompletionProc

A pointer to a completion procedure for asynchronous operations. The calling **tween** component should set the value of this parameter to NIL.

transferProc

A pointer to a procedure to transfer the data. The calling **tween** component should set the value of this parameter to NIL.

refCon

A pointer to a reference constant. The calling **tween** component should set the value of this parameter to NIL.

function result See “Error Codes” (IV–2663). Your callback should return noErr if there is no error.

See Also

See the TweenRecord (IV–2493) and TweenV1Record (IV–2495) structures and the QTDoTween (II–1165), TweenerDoTween (III–1976), and NewTweenerDataUPP (II–1123) functions.

VdigIntProc

An interrupt callback called by the video digitizer component each time it starts to display a field.

```
typedef void (*VdigIntProcPtr) (long flags, long refcon);
```

```
// Declaration of a typical application-defined function
void MyVdigIntProc (
    long    flags
    long    refcon );
```

flags

Undocumented

refcon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

See Also

See the `VDSetDigitizerUserInterrupt` (III–2039) and `NewVdigIntUPP` (II–1125) functions.

Volume IV

Types and Constants

Data Structures

AliasRecord

Defines a Mac OS **alias** record.

```
struct AliasRecord {
    OSType      userType;
    unsigned short aliasSize;
};
```

userType

A 4-byte field that can contain application-specific data, such as the application's signature. When the **alias** record is created, this field contains 0.

aliasSize

The size, in bytes, assigned to the **alias** record when it is created or updated. The size includes the **userType** and **aliasSize** fields plus additional data that is private to the Mac OS.

Discussion

The two declared fields of the `AliasRecord` structure are followed by a variable-length block of data maintained privately by the Mac OS. `AliasRecord` structures may be used inside of 'dref' atoms to reference other movies.

Version Notes

Originally part of the Mac OS.

Programming Info

C interface file: `Aliases.h`

See Also

The `AliasRecord` structure is created by `QTNewAlias` (II-1202). For information about Mac OS **aliases**, see *Inside Macintosh: Files* (listed in the **bibliography**).

ARGBColor

Specifies a color with **alpha**, red, green, and blue values.

```
struct ARGBColor {
    unsigned short  alpha;
    unsigned short  red;
```

AudioInfo

```
        unsigned short    green;  
        unsigned short    blue;  
};
```

alpha
Specifies the alpha component of the color.

red
Specifies the red component of the color.

green
Specifies the green component of the color.

blue
Specifies the blue component of the color.

Discussion

This structure is used in QuickTime vector graphics and other other graphics routines.

Programming Info

C interface file: ImageCodec.h

AudioInfo

Deprecated. Stores information for AudioGetInfo (I–48).

```
struct AudioInfo {  
    long        capabilitiesFlags;  
    long        reserved;  
    unsigned short    numVolumeSteps;  
};
```

capabilitiesFlags
Describes a device’s capabilities; see “Sound Device Features” (IV–2692).

reserved
Reserved by Apple.

numVolumeSteps
The number of significant increments between minimum and maximum volume.

Associated Functions

AudioGetInfo (I–48)

Programming Info

C interface file: Sound.h

AudioSelection

Deprecated.

```
struct AudioSelection {
    long          unitType;
    UnsignedFixed selStart;
    UnsignedFixed selEnd;
};
```

Associated Functions

SndStartFilePlay (III-1847)

Programming Info

C interface file: Sound.h

BandwidthManagementPrefsRecord

Undocumented

```
struct BandwidthManagementPrefsRecord {
    Boolean    overrideConnectionSpeedForBandwidth;
};
```

overrideConnectionSpeedForBandwidth

*Undocumented***Programming Info**

C interface file: Movies.h

BigEndianFixed

Protects a **big-endian** Fixed value from being changed by **little-endian** code.

```
struct BigEndianFixed {
    Fixed    bigEndianValue;
};
```

BigEndianLong

bigEndianValue
A Fixed value.

Special Considerations

On Windows, BigEndianFixed is defined as a struct that contains a fixed variable, which causes a compiler error if you attempt to directly access the variable. In effect, the error serves as a reminder to endian-flip the variable.

Programming Info

C interface file: Endian.h

BigEndianLong

Protects a **big-endian** long value from being changed by **little-endian** code.

```
struct BigEndianLong {  
    long    bigEndianValue;  
};
```

bigEndianValue
A long value.

Programming Info

C interface file: Endian.h

BigEndianOSType

Protects a **big-endian** OSType value from being changed by **little-endian** code.

```
struct BigEndianOSType {  
    OSType    bigEndianValue;  
};
```

bigEndianValue
An OSType value.

Programming Info

C interface file: Endian.h

BigEndianShort

Protects a **big-endian** short value from being changed by **little-endian** code.


```
struct BigEndianShort {  
    short    bigEndianValue;  
};
```

bigEndianValue
A short value.

Programming Info

C interface file: Endian.h

BigEndianUnsignedFixed

Protects a **big-endian** unsigned Fixed value from being changed by **little-endian** code.

```
struct BigEndianUnsignedFixed {  
    UnsignedFixed    bigEndianValue;  
};
```

bigEndianValue
An unsigned Fixed value.

Programming Info

C interface file: Endian.h

BigEndianUnsignedLong

Protects a **big-endian** unsigned long value from being changed by **little-endian** code.

```
struct BigEndianUnsignedLong {  
    unsigned long    bigEndianValue;  
};
```

bigEndianValue
An unsigned long value.

Programming Info

C interface file: Endian.h

BigEndianUnsignedShort

BigEndianUnsignedShort

Protects a **big-endian** unsigned short value from being changed by **little-endian** code.

```
struct BigEndianUnsignedShort {  
    unsigned short    bigEndianValue;  
};
```

bigEndianValue

An unsigned short value.

Programming Info

C interface file: Endian.h

BitMap

Defines a bit image using Macintosh QuickDraw coordinates.

```
struct BitMap {  
    Ptr    baseAddr;  
    short  rowBytes;  
    Rect   bounds;  
};
```

baseAddr

A pointer to the beginning of the bit image.

rowBytes

The offset in bytes from one row of the image to the next. This value may not exceed 0x4000.

bounds

The bitmap's boundary rectangle; by default, the whole computer screen.

Discussion

The width passed in the bounds parameter determines how many bits of one row are part of the bitmap. This value must not exceed the number of bits in each row of the bit image. The height passed in the bounds parameter determines how many rows are part of the bitmap. This value must not exceed the number of rows in the bit image. The rectangle passed in the bounds parameter defines the local coordinate system used by the portRect field of the GrafPort (IV-2273) structure.

Associated Functions

QDBitsProc (III-2113)

Programming Info

C interface file: Quickdraw.h

See Also

See the Rect (IV–2399) and Pixmap (IV–2332) structures.

BooleanRangeRecord

Undocumented

```
struct BooleanRangeRecord {
    long    maskValue;
};
```

maskValue

Undocumented

Discussion

Undocumented

Programming Info

C interface file: ImageCodec.h

CallbackRecord

Stores data for a QTCallbackProc (III–2124).

```
struct CallbackRecord {
    long    data[1];
};
```

data

Callback data.

Programming Info

C interface file: Movies.h

CDSequenceDataSource

Contains a linked list of all data sources for a decompression sequence.

```
struct CDSequenceDataSource {
    long                                recordSize;
```

CDSequenceDataSource

```
void *          next;
ImageSequence   seqID;
ImageSequenceDataSource sourceID;
OSType          sourceType;
long            sourceInputNumber;
void *          dataPtr;
Handle          dataDescription;
long            changeSeed;
ICMConvertDataFormatUPP transferProc;
void *          transferRefcon;
long            dataSize;
QHdrPtr         dataQueue;
void *          originalDataPtr;
long            originalDataSize;
Handle          originalDataDescription;
long            originalDataDescriptionSeed;
};
```

recordSize

The size of this structure.

next

A pointer to the next source entry. If it is NIL, there are no more entries.

seqID

The image sequence that this source is associated with.

sourceID

The source reference identifying this source.

sourceType

A four-character code describing how the input will be used. This value is passed to this parameter by CDSequenceNewDataSource (I-82) when the source is created.

sourceInputNumber

A value is passed to this parameter by CDSequenceNewDataSource when the source is created.

dataPtr

A pointer to the actual source data.

dataDescription

A handle to a data structure describing the data format. This is often a handle to an ImageDescription (IV-2285) structure.

changeSeed

An integer that is incremented each time the dataPtr field changes or that data that the dataPtr field points to changes. By remembering the value of this field

and comparing to the value the next time the decompressor or compressor component is called, the component can determine if new data is present.

transferProc

Reserved.

transferRefcon

Reserved.

dataSize

The size of the data pointed to by the dataPtr field.

dataQueue

A pointer to a QHdr (IV–2345) structure that contains a queue of CDSequenceDataSourceQueueEntry (IV–2167) structures.

originalDataPtr

The original value of dataPtr.

originalDataSize

The original value of dataSize.

originalDataDescription

The original value of dataDescription.

originalDataDescriptionSeed

The original value of changeSeed.

Discussion

Because each data source is associated with a link to the next data source, a codec can access all data sources using this structure.

Version Notes

Fields from dataQueue onward were introduced in QuickTime 3.

Associated Functions

ImageCodecGetMaxCompressionSizeWithSources (II–716)

Programming Info

C interface file: ImageCodec.h

See Also

See CDSequenceDataSourceQueueEntry (IV–2167).

CDSequenceDataSourceQueueEntry

Provides data for CDSequenceDataSource (IV–2165).

```
struct CDSequenceDataSourceQueueEntry {
```

CGrafPort

```
void *      nextBusy;
long       descSeed;
Handle     dataDesc;
void *      data;
long       dataSize;
long       useCount;
TimeValue  frameTime;
TimeValue  frameDuration;
TimeValue  timeScale;
};
```

nextBusy

Undocumented

descSeed

Undocumented

dataDesc

Undocumented

data

Undocumented

dataSize

Undocumented

useCount

Undocumented

frameTime

Undocumented

frameDuration

Undocumented

timeScale

Undocumented

Programming Info

C interface file: ImageCodec.h

See Also

See CDSequenceDataSource (IV–2165).

CGrafPort

Defines a complete drawing environment for color graphics operations.

```
struct CGrafPort {
```

```

short          device;
PixMapHandle   portPixMap;
short          portVersion;
Handle         grafVars;
short          chExtra;
short          pnLochFrac;
Rect           portRect;
RgnHandle      visRgn;
RgnHandle      clipRgn;
PixPatHandle   bkPixPat;
RGBColor       rgbFgColor;
RGBColor       rgbBkColor;
Point          pnLoc;
Point          pnSize;
short          pnMode;
PixPatHandle   pnPixPat;
PixPatHandle   fillPixPat;
short          pnVis;
short          txFont;
StyleField     txFace;
short          txMode;
short          txSize;
Fixed          spExtra;
long           fgColor;
long           bkColor;
short          colrBit;
short          patStretch;
Handle         picSave;
Handle         rgnSave;
Handle         polySave;
CQDProcsPtr    grafProcs;
};

```

device

Device-specific information that QuickDraw uses to achieve the best possible results when drawing text in the graphics port. There may be physical differences in the same logical font for different output devices, to ensure the highest-quality printing on the device being used. The default value of the device field is 0, indicating the computer screen.

portPixMap

A handle to a `PixMap` (IV-2332) structure, which describes the pixels in this color graphics port.

portVersion

The highest 2 bits are permanently set to indicate that this is a `CGrafPort` structure and the remainder of the field contains the version number of

CGrafPort

Macintosh Color QuickDraw that created this structure. Currently initialized to 0xC000.

grafVars

A handle to a GrafVars structure that contains additional graphics fields of color information. On initialization, black is assigned to the rgbOpColor field of this structure, the default highlight color is assigned to the rgbHiLiteColor field, and all other fields are set to 0. For information about the GrafVars structure, see *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

chExtra

A number by which to widen every character, excluding spaces, in a line of text. This value is used in proportional spacing. The value in this field is in 4.12 fractional notation: 4 bits of signed integer followed by 12 bits of fraction. This value is multiplied by the value in the txSize field before it is used. By default, this field contains 0.

pnLocHFrac

The fractional horizontal pen position used when drawing text. The value in this field represents the low word of type Fixed; in decimal, its initial value is 0.5.

portRect

The port rectangle that defines a subset of the pixel map to be used for drawing. All drawing done by the application occurs inside the port rectangle. The port rectangle (also called the content region) uses the local coordinate system defined by the boundary rectangle in the portPixMap field of the PixMap structure. The upper-left corner (which for a window is called the window origin) of the port rectangle usually has vertical and horizontal coordinates of 0. The port rectangle usually falls within the boundary rectangle, but it's not required to do so.

visRgn

The region of the graphics port that's actually visible on the screen; that is, the part of the window that's not covered by other windows. By default, the visible region is equivalent to the port rectangle. The visible region has no effect on images that aren't displayed on the screen.

clipRgn

A handle to the graphics port's clipping region, an arbitrary region that you can use to limit drawing to any region within the port rectangle. Unlike the visible region, the clipping region affects the image even if it isn't displayed on the screen. Initially the clip region is set to the rectangle -32768, -32768, 32767, 32767.

bkPixPat

A handle to a `PixPat` (IV–2336) structure that describes the background pixel pattern, initially set to white.

rgbFgColor

An `RGBColor` (IV–2403) structure that defines the requested foreground color. By default, the foreground color is black.

rgbBkColor

An `RGBColor` (IV–2403) structure that defines the requested background color. By default, the background color is white.

pnLoc

The point where QuickTime will begin drawing the next line, shape, or character. It can be anywhere on the coordinate plane; there are no restrictions on the movement or placement of the pen. The location of the graphics pen is a point in the graphics port's coordinate system, not a pixel in a pixel image. The upper-left corner of the pen is at the pen location; the graphics pen hangs below and to the right of this point. This field is initialized to 0,0.

pnSize

The vertical height and horizontal width of the graphics pen. The default size is a 1-by-1 pixel square; the vertical height and horizontal width can range from 0 by 0 to 32,767 by 32,767. If either the pen width or the pen height is 0, the pen does not draw. Heights or widths of less than 0 are undefined.

pnMode

The pattern mode, a `Boolean` operation that determines the how QuickTime transfers the pen pattern to the pixel map during drawing operations. See “Graphics Transfer Modes” (IV–2670). When the graphics pen draws into a pixel map, QuickTime first determines what pixels in the pixel image are affected and finds their corresponding pixels in the pen pattern. It then does a pixel-by-pixel comparison based on the pattern mode, which specifies one of eight `Boolean` transfer operations to perform. QuickTime stores the resulting pixel in its proper place in the image. This field is initially set to `patCopy`.

pnPixPat

A handle to a `PixPat` (IV–2336) structure that describes a pixel pattern that can be used like the ink in the graphics pen. This field is initially set to black.

fillPixPat

A handle to a `PixPat` (IV–2336) structure that describes the pixel pattern that's used to fill an area. This field is initially set to black. Notice that this is not in the same location as the `fillPat` field in the `GrafPort` (IV–2273) structure.

CGrafPort

pnVis

The graphics pen's visibility; that is, whether or not it draws on the screen. This field is initially set to 0 (visible).

txFont

A font number that identifies the font to be used in the graphics port. This field is initially set to 0, indicating the system font.

txFace

The character style of the text, with values from the set defined by the Style type, which includes such styles as bold, italic, and shaded. You can apply stylistic variations either alone or in combination. This field is initially set to plain text.

txMode

One of three Boolean source mode constants (see below) that determines the way characters are placed in the bit image. This mode functions much like a pattern mode specified in the pnMode field; when drawing a character, QuickTime determines which pixels in the image are affected, does a pixel-by-pixel comparison based on the mode, and stores the resulting pixels in the image. This field is initially set to srcOr.

txSize

The text size in pixels. QuickTime uses this information to provide the bitmaps for text drawing. The txSize value can be represented by the formula (size in points) x (device resolution) / 72 dpi. This field is initially set to the system font size.

spExtra

A number equal to the average number of pixels by which each space character should be widened to fill out a fully justified text line. This field is useful when a line of characters is to be aligned with both the left and the right margin. This field is initially set to 0.

fgColor

The pixel value of the foreground color. This is the best available approximation in the **color lookup table** (CLUT) to the color specified in the rgbFgColor field. This field is initially set to blackColor; see “Color Constants” (IV-2657).

bkColor

The pixel value of the background color. This is the best available approximation in the **color lookup table** (CLUT) to the color specified in the rgbBkColor field. This field is initially set to whiteColor; see “Color Constants” (IV-2657).

colrBit

Reserved and set to 0.

patStretch

A value, initially set to 0, used during output to a printer to expand patterns if necessary. Your application should not change this value.

picSave

A handle to the state of the Macintosh picture definition. If no picture is open, this field contains NIL; otherwise it contains a handle to information related to the picture definition. Your application shouldn't be concerned about exactly what information the handle leads to; you may, however, save the current value of this field, set the field to NIL to disable the picture definition, and later restore it to the saved value to resume defining the picture.

rgnSave

A handle to the state of the region definition. If no region is open, this field contains NIL; otherwise it contains a handle to information related to the region definition. Your application shouldn't be concerned about exactly what information the handle leads to; you may, however, save the current value of this field, set the field to NIL to disable the region definition, and later restore it to the saved value to resume defining the region.

polySave

A handle to the state of the polygon definition. If no polygon is open, this field contains NIL; otherwise it contains a handle to information related to the polygon definition. Your application shouldn't be concerned about exactly what information the handle leads to; you may, however, save the current value of this field, set the field to NIL to disable the polygon definition, and later restore it to the saved value to resume defining the polygon.

grafProcs

An optional pointer to a CQDProcs (IV-2226) structure that your application can store into if you want to customize Color QuickDraw drawing routines or use Color QuickDraw in other advanced, highly specialized ways. This field is initially set to NIL.

txMode Constants

srcOr

If the source pixel is black, apply the foreground color. If the source pixel is white, leave it white. If the source pixel is any other color, apply weighted portions of foreground color.

srcXor

If the source pixel is black, invert the color (undefined for a colored destination pixel). If the source pixel is white or any other color, leave the color as is.

CleanApertureImageDescriptionExtension

srcBic

If the source pixel is black, apply the background color. If the source pixel is white, leave the color as is. If the source pixel is any other color, apply weighted portions of the background color.

Discussion

You can have many graphics ports open at once; each one has its own local coordinate system, pen pattern, background pattern, pen size and location, font and font style, and pixel map in which drawing takes place. Several fields in this structure define your application's drawing area. All drawing in a graphics port occurs in the intersection of the graphics port's boundary rectangle and its port rectangle. Within that intersection, all drawing is cropped to the graphics port's visible region and its clipping region.

Special Considerations

Before a color graphics port defined by `CGrafPort` can be used, it must be allocated and initialized. Once a `CGrafPort` is initialized you can read its fields directly, but you should not store values directly into them. You must use QuickTime functions to alter the contents of a `CGrafPort` structure.

Version Notes

The `CGrafPort` structure supersedes the earlier `GrafPort` (IV–2273) structure.

Programming Info

C interface file: `Quickdraw.h`

CleanApertureImageDescriptionExtension

An extension to `ImageDescription` (IV–2285) that describes the dimensions of a clean aperture.

```
struct CleanApertureImageDescriptionExtension {
    UInt32    cleanApertureWidthN;
    UInt32    cleanApertureWidthD;
    UInt32    cleanApertureHeightN;
    UInt32    cleanApertureHeightD;
    UInt32    horizOffN;
    UInt32    horizOffD;
    UInt32    vertOffN;
    UInt32    vertOffD;
};
```

`cleanApertureWidthN`
Width numerator.

`cleanApertureWidthD`

Width denominator.

`cleanApertureHeightN`

Height numerator.

`cleanApertureHeightD`

Height denominator

`horizOffN`

Numerator of horizontal offset of clean aperture center minus $(width-1)/2$.

`horizOffD`

Denominator of horizontal offset of clean aperture center minus $(width-1)/2$.

`vertOffN`

Numerator of vertical offset of clean aperture center minus $(height-1)/2$.

`vertOffD`

Denominator of vertical offset of clean aperture center minus $(height-1)/2$.

Programming Info

C interface file: `ImageCompression.h`

See Also

See `ImageDescription` (IV-2285).

CloneRecord

Defines a track clone.

```
struct CloneRecord {
    long    flags;
    long    masterTrackID;
};
```

`flags`

Currently 0.

`masterTrackID`

The track ID of the track being cloned.

Programming Info

C interface file: `MoviesFormat.h`

See Also

See the 'clon' ('-clon') atom and `AddClonedTrackToMovie` (I-22).

CmpSoundHeader

CmpSoundHeader

Describes compressed sampled-sound data.

```
struct CmpSoundHeader {
    Ptr          samplePtr;
    unsigned long numChannels;
    UnsignedFixed sampleRate;
    unsigned long loopStart;
    unsigned long loopEnd;
    UInt8        encode;
    UInt8        baseFrequency;
    unsigned long numFrames;
    extended80   AIFFSampleRate;
    Ptr          markerChunk;
    OSType       format;
    unsigned long futureUse2;
    StateBlockPtr stateVars;
    LeftOverBlockPtr leftOverSamples;
    short        compressionID;
    unsigned short packetSize;
    unsigned short snthID;
    unsigned short sampleSize;
    UInt8        sampleArea[1];
};
```

samplePtr

The location of the compressed sound frames. If `samplePtr` is NIL, then the frames are located in the `sampleArea` field of the compressed sound header. Otherwise, `samplePtr` points to a buffer that contains the frames.

numChannels

The number of channels in the sample.

sampleRate

The sample rate at which the frames were sampled before compression; see “Sound Sample Rates” (IV–2695).

loopStart

The beginning of the loop points of the sound before compression. The loop starting and ending points are 0-based.

loopEnd

The end of the loop points of the sound before compression.

encode

The method of encoding (if any) used to generate the sampled-sound data (see below). For a compressed sound header, you should specify the constant `cmpSH`.

Encode option values in the ranges 0 through 63 and 128 to 255 are reserved for use by Apple. You are free to use numbers in the range 64 through 127 for your own encode options.

`baseFrequency`

The pitch of the original sampled sound.

`numFrames`

The number of frames contained in the compressed sound header. When you store multiple channels of noncompressed sound, store them as interleaved sample frames (as in AIFF). When you store multiple channels of compressed sounds, store them as interleaved packet frames.

`AIFFSampleRate`

The sample rate at which the frames were sampled before compression, as expressed in the 80-bit extended data type representation (IEEE sample rate).

`markerChunk`

Synchronization information. The `markerChunk` field is not presently used and should be set to `NIL`.

`format`

The data format type; see “Sound Formats” (IV–2692). This field contains a value of type `OSType` that defines the compression algorithm, if any, used to generate the audio data. For example, for data generated using **MACE 3:1** compression, this field should contain the value `'MAC3'`. This field is used only if the `compressionID` field contains the value `fixedCompression`.

`futureUse2`

This field is reserved for use by Apple. To maintain compatibility with future versions of system software, you should always set this field to 0.

`stateVars`

A pointer to a `StateBlock` (IV–2463) structure. This field is used to store the state variables for a given algorithm across consecutive calls.

`leftOverSamples`

A pointer to a `LeftOverBlock` (IV–2301) structure. You can use this block to store samples that will be truncated across algorithm invocations.

`compressionID`

The compression algorithm used on the samples in the compressed sound header (see below). The constant `fixedCompression` is available only with Sound Manager versions 3.0 and later. If the `compressionID` field contains the value `fixedCompression`, the Sound Manager reads the `format` field to determine the compression algorithm used to generate the compressed data. Otherwise, the Sound Manager reads the `compressionID` field. Apple reserves the right to use

CmpSoundHeader

compression IDs in the range 0 through 511. Currently the constant `variableCompression` is not used by the Sound Manager.

`packetSize`

The size, in bits, of the smallest element that a given expansion algorithm can work with (see below). Beginning with Sound Manager version 3.0, you can specify the value 0 in this field to instruct the Sound Manager to determine the packet size itself.

`snthID`

Resource ID of the Sound Manager synthesizer that contains NRT compression/expansion. If this field is not used, set it to 0.

`sampleSize`

The size of the sample before it was compressed. The samples passed in the compressed sound header should always be byte-aligned, and any padding done to achieve byte alignment should be done from the left with zeros.

`sampleArea`

The sample frames, but only when the `samplePtr` field is `NIL`. Otherwise, the sample frames are in the location indicated by `samplePtr`.

encode Constants

`cmpSH`

Compressed sound header; value is 0xFE.

`extSH`

Extended sound header; value is 0xFF.

`stdSH`

Standard sound header; value is 0x00.

compressionID Constants

`variableCompression`

Variable-ratio compression; value is -2.

`fixedCompression`

Fixed-ratio compression; value is -1.

`notCompressed`

Uncompressed samples; value is 0.

`threeToOne`

3:1 compressed samples; value is 3.

`sixToOne`

6:1 compressed samples; value is 4

packetSize Constants

sixToOnePacketSize

Size for 6:1 compression; value is 8.

threeToOnePacketSize

Size for 3:1 compression; value is 16.

Discussion

Compressed sound headers include all of the essential fields of `ExtSoundHeader` (IV–2255) in addition to several fields that pertain to compression.

Programming Info

C interface file: `Sound.h`

See Also

See `ExtSoundHeader` (IV–2255).

CodecCapabilities

Lets image compressor components report their capabilities to the Image Compression Manager.

```
struct CodecCapabilities {
    long          flags;
    short         wantedPixelSize;
    short         extendWidth;
    short         extendHeight;
    short         bandMin;
    short         bandInc;
    short         pad;
    unsigned long time;
    long          flags2;
};
```

`flags`

Contains flags (see below) that contain control information used by both the Image Compression Manager and the compressor component.

`wantedPixelSize`

Indicates the pixel depth the component can use with the specified image. The component determines the pixel depth of the image for the operation by examining the appropriate pixel map.

`extendWidth`

Specifies the number of pixels the image must be extended in width. If the component cannot accommodate the image at its given width, the component

CodecCapabilities

may request that the Image Compression Manager extend the width of the image by adding pixels to the right edge of the image. This is sometimes necessary to accommodate the component's block size.

`extendHeight`

Specifies the number of pixels the image must be extended in height. If the component cannot accommodate the image at its given height the component may request that the Image Compression Manager extend the height of the image by adding pixels to the bottom of the image. This is sometimes necessary to accommodate the component's block size.

`bandMin`

Contains the minimum image **band** height supported by the component. Components that can tolerate small values operate under a wider set of memory conditions.

`bandInc`

Specifies a common factor of supported image **band** heights. A component may support only image bands that are an even multiple of some number of pixels high. These components report this common factor in the `bandInc` field. Set this field to 1 if your component supports bands of any size.

`pad`

Reserved for use by Apple.

`time`

Indicates the number of milliseconds the operation will take to complete. If the compressor cannot determine this value, it sets this field to 0.

`flags2`

Additional compressor capability flags (see below).

flags Constants

`codecCanScale`

Indicates whether the decompressor can scale the image during decompression. The decompressor sets this flag to 1 to indicate that it can scale the image during decompression. The decompressor sets this flag to 0 if it cannot scale the decompressed image.

`codecCanMask`

Indicates whether the decompressor can apply a mask to the decompressed image. The decompressor sets this flag to 1 to indicate that it can use a mask to control the image that results from a decompression operation. The decompressor sets this flag to 0 if it cannot work with masks.

`codecCanMatte`

Indicates whether the decompressor can blend the decompressed image using a matte. The decompressor sets this flag to 1 to indicate that it can use a blend matte during decompression. The decompressor sets this flag to 0 if it cannot use a blend matte.

`codecCanTransform`

Indicates whether the decompressor can work with complex placement matrixes. The decompressor sets this flag to 1 to indicate that it can work with transformation matrixes during decompression. The decompressor sets this flag to 0 if it cannot work with matrixes.

`codecCanTransferMode`

Indicates whether the decompressor can accept a transfer mode other than source copy or **dither** copy when displaying a decompressed image. The decompressor sets this flag to 1 to indicate that it can accept transfer modes; otherwise, the decompressor sets this flag to 0.

`codecCanCopyPrev`

Indicates whether the compressor can update the previous image buffer during sequence compression. The compressor sets this flag to 1 to indicate that it can update the previous image buffer. The compressor sets this flag to 0 if it cannot update the buffer.

`codecCanSpool`

Indicates whether the component can use data-loading and data-unloading functions to spool data during decompression and compression operations, respectively. Applications may define data-loading and data-unloading functions to handle images that cannot fit into memory. See the chapter “Image Compression Manager” in *Inside Macintosh: QuickTime* (listed in the **bibliography**) for more information on data-loading and data-unloading functions. The component sets this flag to 1 to indicate that it can use these functions. The component sets this flag to 0 to indicate that it cannot use these functions.

`codecCanClipVertical`

Indicates whether the decompressor can clip an image vertically during decompression. The decompressor sets this flag to 1 to indicate that it can clip an image vertically. The decompressor sets this flag to 0 to indicate that it cannot clip an image vertically.

`codecCanClipRectangular`

Indicates whether the decompressor can clip both vertically and horizontally during decompression. The decompressor sets this flag to 1 to indicate that it

CodecCapabilities

can clip along both axes. The decompressor sets this flag to 0 to indicate that it cannot clip an image both vertically and horizontally.

`codecCanRemapColor`

Indicates whether the compressor can remap the colors for an image using color tables. The compressor sets this flag to 1 if it can remap colors. The compressor sets this flag to 0 if it cannot remap colors.

`codecCanFastDither`

Indicates whether the compressor supports fast **dithering**. The compressor sets this flag to 1 if it supports fast dithering. The compressor sets this flag to 0 if it does not support fast dithering. See the chapter “Image Compression Manager” in *Inside Macintosh: QuickTime* (listed in the **bibliography**) for more information about fast dithering.

`codecCanSrcExtract`

Indicates whether the compressor can extract a portion of the source image. The compressor sets this flag to 1 if it can extract a portion of the source image. The compressor sets the flag to 0 if it cannot.

`codecCanCopyPrevComp`

Indicates whether the compressor can update the previous image buffer during sequence compression using compressed data. The compressor sets this flag to 1 to indicate that it can update the previous image buffer. The compressor sets this flag to 0 if it cannot update the buffer.

`codecCanAsync`

Indicates whether the component can work asynchronously. The compressor sets this flag to 1 if it can compress and decompress asynchronously; otherwise, it sets this flag to 0.

`codecCanMakeMask`

Indicates whether the decompressor creates modification masks during decompression. These masks indicate which pixels in the decompressed image differ from the previous image and must therefore be displayed. Such masks are useful only when processing sequences. The decompressor sets this flag to 1 to indicate that it creates modification masks. The decompressor sets this flag to 0 if it does not create such masks.

`codecCanShift`

Indicates whether the component can work with pixels that are not byte-aligned. This flag is valid only when the source or destination uses fewer than 8 bits per pixel. Components set this flag to 1 if they can read or write pixels that are not byte-aligned. Components set this flag to 0 if pixels must be byte-aligned.

flags2 Constants**codecCanAsyncWhen**

Indicates whether your decompressor component supports scheduled asynchronous decompression. Set this flag to 1 if your component can support the scheduling functionality of `ImageCodecBandDecompress` (II-689). Note that you must also set the `codecCanAsync` flag to 1.

codecCanShieldCursor

Indicates whether your decompressor component will manage the shielding of the cursor during decompression. If your component can manage the cursor's display, set this flag to 1. Your component can use `ICMSHieldSequenceCursor` (II-679) to shield the cursor. The cursor is automatically unshielded when you call `ICMDecompressComplete` (II-671). Otherwise, set this flag to 0; the Image Compression Manager then manages the cursor for you. It is highly recommended that you support this capability if your decompressor supports asynchronous operation or the cursor may remain shielded for unacceptably long periods of time.

codecCanManagePrevBuffer

Indicates that your compressor component is capable of allocating and managing the `prevPixMap` used in temporal compression. If this flag is set, then your compressor must determine when to update the `prevPixMap` during compression sequences. Codecs setting this flag should also set `codecCanCopyPrev`.

codecHasVolatileBuffer

Some hardware decompressors don't actually draw the decompressed pixels into the frame buffer as requested by QuickTime. Instead, they have a second frame buffer that floats or overlays above the main frame buffer. The image is decompressed into this secondary frame buffer instead. To the user, the effect is the same because the video hardware merges the two frame buffers together. When the window that contains the image is moved to another location in the same screen buffer, the Window Manager uses `CopyBits` to transfer the window's pixels from the old location to the new location. Unfortunately, because the Window Manager is unaware of the presence of the secondary frame buffer, it cannot move the image it is displaying. By setting the `codecHasVolatileBuffer` flag to 1, the decompressor component informs QuickTime that it uses a secondary frame buffer. When the Window Manager moves a window, QuickTime forces a redraw of the contents of the window so that the secondary frame buffer can be repositioned and / or updated as necessary.

CodecCompressParams

codecImageBufferIsOnScreen

By setting the `codecImageBufferIsOnScreen` flag to 1, the decompressor component informs QuickTime that it is a direct screen transfer codec. Codecs that use `ImageCodecNewImageBufferMemory` (II–727) to create an offscreen buffer that is really onscreen would set this flag.

codecWantsSpecialScaling

Specifies to use an image buffer whose size is determined by the `requestedBufferWidth` and `requestedBufferHeight` fields in `CodecDecompressParams` (IV–2190) or `CodecCompressParams` (IV–2184).

codecWantsDestinationPixels

Undocumented

Discussion

Before compressing or decompressing an image, the Image Compression Manager requests capability information from the component that will be handling the operation by calling `ImageCodecPreCompress` (II–730) or `ImageCodecPreDecompress` (II–731). The compressor component examines the compression or decompression parameters and indicates any restrictions on its ability to satisfy the request in a formatted compressor capability structure. The Image Compression Manager then manages the operation according to the capabilities of the component.

Programming Info

C interface file: `ImageCodec.h`

CodecCompressParams

Contains parameters that govern a compression operation.

```
struct CodecCompressParams {
    ImageSequence          sequenceID;
    ImageDescriptionHandle imageDescription;
    Ptr                    data;
    long                   bufferSize;
    long                   frameNumber;
    long                   startLine;
    long                   stopLine;
    long                   conditionFlags;
    CodecFlags             callerFlags;
    CodecCapabilities *    capabilities;
    ICMProgressProcRecord  progressProcRecord;
    ICMCompletionProcRecord completionProcRecord;
    ICMFlushProcRecord     flushProcRecord;
    PixMap                 srcPixMap;
```

```

    PixMap                prevPixMap;
    CodecQ                spatialQuality;
    CodecQ                temporalQuality;
    Fixed                 similarity;
    DataRateParamsPtr     dataRateParams;
    long                  reserved;
    UInt16                majorSourceChangeSeed;
    UInt16                minorSourceChangeSeed;
    CDSequenceDataSourcePtr sourceData;
    long                  preferredPacketSizeInBytes;
    long                  requestedBufferWidth;
    long                  requestedBufferHeight;
    OSType                wantedSourcePixelFormat;
    long                  compressedDataSize;
    UInt32                taskWeight;
    OSType                taskName;
};

```

sequenceID

Contains a unique sequence identifier. If the image to be compressed is part of a sequence, this field contains the sequence identifier that was assigned by `CompressSequenceBegin` (I–119). If the image is not part of a sequence, this field is set to 0.

imageDescription

Contains a handle to the image description structure that describes the image to be compressed.

data

Points to a location to receive the compressed image data. This is a **32-bit clean** address. If there is not sufficient memory to store the compressed image, the application may choose to write the compressed data to mass storage during the compression operation. The `flushProcRecord` field identifies the data-unloading function that the application provides for this purpose. This field is used only by `ImageCodecBandCompress` (II–688).

bufferSize

Contains the size of the buffer specified by the `data` field. Your component sets the value of the `bufferSize` field to the number of bytes of compressed data written into the buffer. Your component should not return more data than the buffer can hold; it should return a nonzero result code instead. This field is used only by `ImageCodecBandCompress` (II–688).

frameNumber

Contains a frame identifier. Indicates the relative frame number within the sequence. The Image Compression Manager increments this value for each

CodecCompressParams

frame in the sequence. This field is used only by `ImageCodecBandCompress` (II-688).

startLine

Contains the starting line for the **band**. This field indicates the starting line number for the band to be compressed. The line number refers to the pixel row in the image, starting from the top of the image. The first row is row number 0. This field is used only by `ImageCodecBandCompress` (II-688).

stopLine

Contains the ending line for the **band**. This field indicates the ending line number for the band to be compressed. The line number refers to the pixel row in the image, starting from the top of the image. The first row in the image is row number 0. The image band includes the row specified by this field. So, to define a band that contains one row of pixels at the top of an image, you set the `startLine` field to 0 and the `stopLine` field to 1.

conditionFlags

Contains flags (see below) that identify the condition under which your component has been called. This field is used only by `ImageCodecBandCompress` (II-688). In addition, these fields contain information about actions taken by your component.

callerFlags

Flags that provide further control information. This field is used only by `ImageCodecBandCompress` (II-688).

capabilities

Points to a compressor capability structure. The Image Compression Manager uses this field to determine the capabilities of your compressor component. This field is used only by `ImageCodecPreCompress` (II-730).

progressProcRecord

Contains an `ICMProgressProcRecord` (IV-2284) structure. During the compression operation, your compressor may occasionally call a function that the application provides in order to report your progress. This field contains a structure that identifies the **progress function**. If the `progressProc` field in this structure is set to `NIL`, the application has not supplied a progress function. This field is used only by `ImageCodecBandCompress` (II-688).

completionProcRecord

Contains an `ICMCompletionProcRecord` (IV-2279) structure. This structure governs whether you perform the compression asynchronously. If the `completionProc` field in this structure is set to `NIL`, perform the compression synchronously. If this field is not `NIL`, it specifies an application completion function. Perform the compression asynchronously and call that completion

function when your component is finished. If the `completionProc` field in this structure has a value of `-1`, perform the operation asynchronously but do not call the application's completion function. This field is used only by `ImageCodecBandCompress` (II-688).

`flushProcRecord`

Contains an `ICMFlushProcRecord` (IV-2280) structure. If there is not enough memory to store the compressed image, the application may provide a function that unloads some of the compressed data. This field contains a structure that identifies that data-unloading function. If the application did not provide a data-unloading function, the `flushProc` field in this structure is set to `NIL`. In this case, your component writes the entire compressed image into the memory location specified by the `data` field. The data-unloading function structure is used only by `ImageCodecBandCompress` (II-688).

`srcPixMap`

Points to the image to be compressed. The image must be stored in a pixel map structure. The contents of this pixel map differ from a standard pixel map in two ways. First, the `rowBytes` field is a full 16-bit value; the high-order bit is not necessarily set to 1. Second, the `baseAddr` field must contain a **32-bit clean** address. This field is used only by `ImageCodecBandCompress` (II-688).

`prevPixMap`

Points to a pixel map containing the previous image. If the image to be compressed is part of a sequence that is being temporally compressed, this field defines the previous image for temporal compression. Your component should then use this previous image as the basis of comparison for the image to be compressed. If the `temporalQuality` field is set to 0, do not perform temporal compression. If the `codecFlagUpdatePrevious` flag or the `codecFlagUpdatePreviousComp` flag in the `flags` field is set to 1, update the previous image at the end of the compression operation. The contents of this pixel map differ from a standard pixel map in two ways. First, the `rowBytes` field is a full 16-bit value; the high-order bit is not necessarily set to 1. Second, the `baseAddr` field must contain a **32-bit clean** address. This field is used only by `ImageCodecBandCompress` (II-688).

`spatialQuality`

Specifies the desired compressed image quality. This field is used only by `ImageCodecBandCompress` (II-688).

`temporalQuality`

Specifies the desired sequence temporal quality. This field governs the level of compression the application desires with respect to information in successive frames in the sequence. If this field is set to 0, do not perform temporal

CodecCompressParams

compression on this frame. This field is used only by `ImageCodecBandCompress` (II-688).

`similarity`

Indicates the similarity between adjacent frames when performing temporal compression. Your component returns a fixed-point number in this field. That value indicates the relative similarity between the frame just compressed and the previous frame. Valid values range from 0 (key frame) to 1 (identical). This field is used only by `ImageCodecBandCompress` (II-688).

`dataRateParams`

Points to the parameters used when performing data rate constraint.

`reserved`

Reserved.

`majorSourceChangeSeed`

Contains an integer value that is incremented each time a data source is added or removed. This provides a fast way for a codec to know when it needs to redetermine which data source inputs are available.

`minorSourceChangeSeed`

Contains an integer value that is incremented each time a data source is added or removed, or the data contained in any of the data sources changes. This provides a way for a codec to know if the data available to it has changed.

`sourceData`

Contains a pointer to a `CDSequenceDataSource` (IV-2165) structure. This structure contains a linked list of all data sources. Because each data source contains a link to the next data source, a codec can access all data sources from this field.

`preferredPacketSizeInBytes`

Specifies the preferred packet size for data.

`requestedBufferWidth`

Specifies the the width of the image buffer to use, in pixels. For this value to be used, the `codecWantsSpecialScaling` flag in the `CodecCapabilities` structure must be set.

`requestedBufferHeight`

Specifies the the height of the image buffer to use, in pixels. For this value to be used, the `codecWantsSpecialScaling` flag in the `CodecCapabilities` structure must be set.

`wantedSourcePixelFormatType`

Undocumented

compressedDataSize

The size of the compressed image, in bytes. If this field is nonzero, it overrides the dataSize field of the ImageDescription (IV-2285) structure. This provides a safer way for asynchronous compressors to return the size of the compressed frame data, because the dataSize field of ImageDescription may be referenced by an unlocked handle.

taskWeight

The preferred weight for multiprocessing tasks implementing this operation. You should assign a value by means of the Mac OS function MPSetTaskWeight.

taskName

The preferred type for multiprocessing tasks implementing this operation. You should assign a value by means of the Mac OS function MPSetTaskType.

conditionFlags Constants

codecConditionFirstBand

An input flag that indicates if this is the first **band** in the frame. If this flag is set to 1, then your component is being called for the first time for the current frame.

codecConditionLastBand

An input flag that indicates if this is the last **band** in the frame. If this flag is set to 1, then your component is being called for the last time for the current frame. If the codecConditionFirstBand flag is also set to 1, this is the only time the Image Compression Manager is calling your component for the current frame.

codecConditionCodecChangedMask

An output flag that indicates that the component has changed the mask bits. If your image decompressor component can mask decompressed images and if some of the image pixels should not be written to the screen, set to 0 the corresponding bits in the mask defined by the maskBits field in the decompression parameter structure. In addition, set this flag to 1. Otherwise, set this flag to 0.

callerFlags Constants

codecFlagUpdatePrevious

Controls whether your compressor updates the previous image during compression. This flag is only used with sequences that are being temporally compressed. If this flag is set to 1, your compressor should copy the current frame into the previous frame buffer at the end of the frame-compression sequence. Use the source image.

codecFlagWasCompressed

Indicates to your compressor that the image to be compressed has been compressed before. This information may be useful to compressors that can compensate for the image degradation that may otherwise result from repeated

CodecDecompressParams

compression and decompression of the same image. This flag is set to 1 to indicate that the image was previously compressed. This flag is set to 0 if the image was not previously compressed.

`codecFlagUpdatePreviousComp`

Controls whether your compressor updates the previous image buffer with the compressed image. This flag is only used with temporal compression. If this flag is set to 1, your compressor should update the previous frame buffer at the end of the frame-compression sequence, allowing your compressor to perform frame differencing against the compression results. Use the image that results from the compression operation. If this flag is set to 0, your compressor should not modify the previous frame buffer during compression.

`codecFlagLiveGrab`

Indicates whether the current sequence results from grabbing live video. When working with live video, your compressor should operate as quickly as possible and disable any additional processing, such as compensation for previously compressed data. This flag is set to 1 when you are compressing from a live video source.

Discussion

Compressor components accept the parameters that govern a compression operation in the form of the `CodecCompressParams` structure. This structure is used by `ImageCodecBandCompress` (II-688) and `ImageCodecPreCompress` (II-730).

Version Notes

Some of the fields in `CodecCompressParams` were added for various versions of QuickTime starting with version 2.1. See comments in the C interface file for details.

Associated Functions

`ImageCodecBandCompress` (II-688)

Programming Info

C interface file: `ImageCodec.h`

See Also

See `ImageCodecBandCompress` (II-688).

CodecDecompressParams

The basic parameter block that is passed to a decompressor.

```
struct CodecDecompressParams {
    ImageSequence          sequenceID;
    ImageDescriptionHandle imageDescription;
    Ptr                    data;
```

```

long                bufferSize;
long                frameNumber;
long                startLine;
long                stopLine;
long                conditionFlags;
CodecFlags          callerFlags;
CodecCapabilities * capabilities;
ICMProgressProcRecord progressProcRecord;
ICMCompletionProcRecord completionProcRecord;
ICMDataProcRecord   dataProcRecord;
CGrafPtr            port;
PixMap              dstPixMap;
BitMapPtr           maskBits;
PixMapPtr           mattePixMap;
Rect                srcRect;
MatrixRecord *      matrix;
CodecQ              accuracy;
short               transferMode;
ICMFrameTimePtr     frameTime;
long                reserved[1];
SInt8               matrixFlags;
SInt8               matrixType;
Rect                dstRect;
UInt16              majorSourceChangeSeed;
UInt16              minorSourceChangeSeed;
CDSequenceDataSourcePtr sourceData;
RgnHandle            maskRegion;
OSType **           wantedDestinationPixelTypes;
long                screenFloodMethod;
long                screenFloodValue;
short               preferredOffscreenPixelSize;
ICMFrameTimeInfoPtr syncFrameTime;
Boolean             needUpdateOnTimeChange;
Boolean             enableBlackLining;
Boolean             needUpdateOnSourceChange;
Boolean             pad;
long                unused;
CGrafPtr            finalDestinationPort;
long                requestedBufferWidth;
long                requestedBufferHeight;
Rect                displayableAreaOfRequestedBuffer;
Boolean             requestedSingleField;
Boolean             needUpdateOnNextIdle;
Boolean             pad2[2];
fixed               bufferGammaLevel;
UInt32              taskWeight;
OSType              taskName;
};

```

CodecDecompressParams

sequenceID

Contains the unique sequence identifier. If the image to be decompressed is part of a sequence, this field contains the sequence identifier that was assigned by `DecompressSequenceBegin` (I–238). If the image is not part of a sequence, this field is set to 0.

imageDescription

Contains a handle to the `ImageDescription` (IV–2285) that describes the image to be decompressed.

data

Points to the compressed image data. This must be a **32-bit clean** address. The `bufferSize` field indicates the size of this data buffer. If the entire compressed image does not fit in memory, the application should provide a data-loading function, identified by the `dataProc` field of the data-loading function structure stored in the `dataProcRecord` field. This field is used only by `ImageCodecBandDecompress` (II–689).

bufferSize

Specifies the size of the image data buffer. This field is used only by `ImageCodecBandDecompress` (II–689).

frameNumber

Contains a frame identifier. Indicates the relative frame number within the sequence. The Image Compression Manager increments this value for each frame in the sequence. This field is used only by `ImageCodecBandDecompress` (II–689).

startLine

Specifies the starting line for the **band**. The line number refers to the pixel row in the image, starting from the top of the image. The first row in the image is row number 0. This field is used only by `ImageCodecBandDecompress` (II–689).

stopLine

Specifies the ending line for the **band**. The line number refers to the pixel row in the image, starting from the top of the image. The first row is row number 0. The image band includes the row specified by this field. So, to define a band that contains one row of pixels at the top of an image, you set the `startLine` field to 0 and the `stopLine` field to 1. This field is used only by `ImageCodecBandDecompress` (II–689).

conditionFlags

Contains flags (see below) that identify the condition under which your component has been called (in order to save the component some work). The flags in this field are passed to the component by `ImageCodecBandCompress` (II–688) and `ImageCodecPreDecompress` (II–731) when conditions change, to save

it some work. In addition, these fields contain information about actions taken by your component.

callerFlags

Contains flags (see below) that provide further control information. This field is used only by ImageCodecBandCompress (II-688).

capabilities

Points to a CodecCapabilities (IV-2179) structure. The Image Compression Manager uses this parameter to determine the capabilities of your decompressor component. This field is used only by ImageCodecPreDecompress (II-731).

progressProcRecord

Contains a ICMProgressProcRecord (IV-2284) structure. During the decompression operation, your decompressor may occasionally call a function that the application provides in order to report your progress. This field contains a structure that identifies the **progress function**. If the progressProc field of this structure is set to NIL, the application did not provide a progress function. This field is used only by ImageCodecBandDecompress (II-689).

completionProcRecord

Contains an ICMCompletionProcRecord (IV-2279) structure. This field governs whether you perform the decompression asynchronously. If the completionProc field in this structure is set to NIL, perform the decompression synchronously. If this field is not NIL, it specifies an application completion function. Perform the decompression asynchronously and call that completion function when your component is finished. If this field has a value of -1, perform the operation asynchronously but do not call the application's completion function. This field is used only by ImageCodecBandDecompress (II-689).

dataProcRecord

Contains an ICMDataProcRecord (IV-2279) structure. If the data stream is not all in memory, your component may call an application function that loads more compressed data. This field contains a structure that identifies that data-loading function. If the application did not provide a data-loading function, the dataProc field in this structure is set to NIL. In this case, the entire image must be in memory at the location specified by the data field. This field is used only by ImageCodecBandDecompress (II-689).

port

Points to the color graphics port that receives the decompressed image.

CodecDecompressParams

dstPixMap

Points to the pixel map where the decompressed image is to be displayed. The `GDevice` global variable is set to the destination graphics device. The contents of this pixel map differ from a standard pixel map in two ways. First, the `rowBytes` field is a full 16-bit value; the high-order bit is not necessarily set to 1. Second, the `baseAddr` field must contain a **32-bit clean** address.

maskBits

Contains an update mask. If your component can mask result data, use this mask to indicate which pixels in the destination pixel map to update. Your component indicates whether it can mask with the `codecCanMask` flag in the `flags` field of the `CodecCapabilities` (IV–2179) structure referred to by the `capabilities` field. This field is updated in response to the `ImageCodecPreDecompress` (II–731) request. If the mask has not changed since the last `ImageCodecBandDecompress` request, the `codecConditionCodecChangedMask` flag in the `conditionFlags` field is set to 0. This field is used only by `ImageCodecBandDecompress` (II–689).

mattePixMap

Points to a pixel map that contains a blend matte. The matte can be defined at any supported pixel depth; the matte depth need not correspond to the source or destination depths. The matte must be in the coordinate system of the source image. If the application does not want to apply a blend matte, this field is set to `NIL`. The contents of this pixel map differ from a standard pixel map in two ways. First, the `rowBytes` field is a full 16-bit value; the high-order bit is not necessarily set to 1. Second, the `baseAddr` field must contain a **32-bit clean** address. This field is used only by `ImageCodecBandDecompress` (II–689).

srcRect

Points to a rectangle defining the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the compressed image, which is defined by the `width` and `height` fields of the image description structure referred to by the `imageDescription` field.

matrix

Points to a matrix structure that specifies how to transform the image during decompression.

accuracy

Constant (see below) that specifies the accuracy desired in the decompressed image. Values for this parameter are on the same scale as compression quality; see `CompressImage` (I–113).

transferMode

Specifies the QuickDraw transfer mode for the operation; see “Graphics Transfer Modes” (IV–2670).

frameTime

Contains a pointer to an ICMFrameTimeRecord (IV–2281) structure. This structure contains a frame’s time information for scheduled asynchronous decompression operations.

matrixFlags

Flag (see below) specifying the transformation matrix. Set to 0 for no transformation.

matrixType

Contains the type of the transformation matrix, as returned by GetMatrixType (I–419).

dstRect

The destination rectangle. It is the result of transforming the source rectangle (the `srcRect` parameter) by the transformation matrix (the `matrix` parameter).

majorSourceChangeSeed

Contains an integer value that is incremented each time a data source is added or removed. This provides a fast way for a codec to know when it needs to redetermine which data source inputs are available.

minorSourceChangeSeed

Contains an integer value that is incremented each time a data source is added or removed, or the data contained in any of the data sources changes. This provides a way for a codec to know if the data available to it has changed.

sourceData

Contains a pointer to a CDSequenceDataSource (IV–2165) structure. This structure contains a linked list of all data sources. Because each data source contains a link to the next data source, a codec can access all data sources from this field.

maskRegion

If the `maskRegion` field is not NIL, it contains a QuickDraw region that is equivalent to the bit map contained in the `maskBits` field. For some codecs, using the QuickDraw region may be more convenient than the mask bit map.

wantedDestinationPixelTypes

Filled in by the codec during the execution of ImageCodecPreDecompress (II–731). Contains a handle to a zero-terminated list of non-RGB pixels that the codec can decompress to. Leave set to NIL if the codec does not support non-RGB pixel spaces. The ICM copies this data structure, so it is up to the codec to dispose of

CodecDecompressParams

it later. Since the predecompress call can be called often, it is suggested that codecs allocate this handle during the Open function and dispose of it during the Close function.

screenFloodMethod

A constant (see below) for codecs that require key-color flooding.

screenFloodValue

If screenFloodMethod is kScreenFloodMethodKeyColor, contains the index of the color that should be used to flood the image area on screen when a refresh occurs. This is valid for both indexed and direct screen devices (e.g., for devices with 16 bit depth, it should contain the 5-5-5 RGB value). If screenFloodMethod is kScreenFloodMethodAlpha, contains the value that the alpha channel should be flooded with.

preferredOffscreenPixelSize

Should be filled in ImageCodecPreDecompress (II-731) with the preferred depth of an offscreen buffer should the ICM have to create one. It is not guaranteed that an offscreen buffer will actually be of this depth. A codec should still be sure to specify what depths it can decompress to by using the capabilities field. A codec might use this field if it was capable of decompressing to several depths, but was faster decompressing to a particular depth.

syncFrameTime

A pointer to an ICMFrameTimeInfo (IV-2281) structure. This structure contains timing information about the display of the frame.

needUpdateOnTimeChange

Undocumented

enableBlackLining

If TRUE, indicates that the client has requested **blacklining** (displaying every other line of the image). Blacklining increases the speed of movie playback while decreasing the image quality.

needUpdateOnSourceChange

Undocumented

pad

Unused.

unused

Unused.

finalDestinationPort

Undocumented

requestedBufferWidth

Specifies the width of the image buffer to use, in pixels. For this value to be used, the `codecWantsSpecialScaling` flag in `CodecCapabilities` (IV-2179) must be set.

requestedBufferHeight

Specifies the height of the image buffer to use, in pixels. For this value to be used, the `codecWantsSpecialScaling` flag in `CodecCapabilities` (IV-2179) must be set.

displayableAreaOfRequestedBuffer

This field can be used to prevent parts of the requested buffer from being displayed. When the `codecWantsSpecialScaling` flag is set, this rectangle can be filled in to indicate what portion of the requested buffer's width and height should be used. The buffer rectangle created by the requested buffer is always based at (0,0), so this coordinate system is also used by `displayableAreaOfRequestedBuffer`. If this field is not filled in, a default value of (0,0,0,0) is used, and the entire buffer is displayed. Use this field if you are experiencing edge problems with FlashPix images.

requestedSingleField

Undocumented

needUpdateOnNextIdle

Undocumented

pad2

Unused.

bufferGammaLevel

The gamma level of the data buffer.

taskWeight

The preferred weight for multiprocessing tasks implementing this operation. You should assign a value by means of the Mac OS function `MPSetTaskWeight`.

taskName

The preferred type for multiprocessing tasks implementing this operation. You should assign a value by means of the Mac OS function `MPSetTaskType`.

conditionFlags Constants

`codecConditionFirstBand`

An input flag that indicates if this is the first **band** in the frame. If this flag is set to 1, then your component is being called for the first time for the current frame.

`codecConditionLastBand`

An input flag that indicates if this is the last **band** in the frame. If this flag is set to 1, then your component is being called for the last time for the current frame.

CodecDecompressParams

If the `codecConditionFirstBand` flag is also set to 1, this is the only time the Image Compression Manager is calling your component for the current frame.

`codecConditionFirstFrame`

An input flag that indicates that this is the first frame to be decompressed for this image sequence.

`codecConditionNewDepth`

An input flag that indicates that the depth of the destination has changed for this image sequence.

`codecConditionNewTransform`

An input flag that indicates that the transformation matrix has changed for this sequence.

`codecConditionNewSrcRect`

An input flag that indicates that the source rectangle has changed for this sequence.

`codecConditionNewMatte`

An input flag that indicates that the matte pixel map has changed for this sequence.

`codecConditionNewTransferMode`

An input flag that indicates that the transfer mode has changed for this sequence.

`codecConditionNewClut`

An input flag that indicates that the **color lookup table** has changed for this sequence.

`codecConditionNewAccuracy`

An input flag that indicates to the component that the accuracy parameter has changed for this sequence.

`codecConditionNewDestination`

An input flag that indicates to the component that the destination pixel map has changed for this sequence.

`codecConditionCodecChangedMask`

An output flag that indicates that the component has changed the mask bits. If your image decompressor component can mask decompressed images and if some of the image pixels should not be written to the screen, set the corresponding bits in the mask (defined by the `maskBits` field) to 0. In addition, set this flag to 1. Otherwise, set this flag to 0.

`codecConditionFirstScreen`

Indicates when the codec is decompressing an image to the first of multiple screens. That is, if the decompressed image crosses multiple screens, then the codec can look at this flag to determine if this is the first time an image is being decompressed for each of the screens to which it is being decompressed. A codec that depends on the `maskBits` field of this structure being a valid `RgnHandle` on `ImageCodecPreDecompress` (II-731) needs to know that in this case it is not able to clip images since the region handle is only passed for the first of the screens; clipping would be incorrect for the subsequent screen for that image.

`codecConditionDoCursor`

Set to 1 if the decompressor component should shield and unshield the cursor for the current decompression operation. This flag should be set only if the codec has indicated its ability to handle cursor shielding by setting the `codecCanShieldCursor` flag in the `capabilities` field during `ImageCodecPreDecompress` (II-731).

`codecConditionCatchUpDiff`

Indicates if the current frame is a “catch-up” frame. Set this flag to 1 if the current frame is a catch-up frame. Note that you must also set the `codecFlagCatchUpDiff` flag to 1. This may be useful to decompressors that can drop frames when playback is falling behind.

`codecConditionMaskMayBeChanged`

The Image Compression Manager has always included support for decompressors that could provide a bit mask of pixels that were actually drawn when a particular frame was decompressed. If a decompressor can provide a bit mask of pixels that changed, the Image Compression Manager transfers to the screen only the pixels that actually changed. QuickTime 2.1 extended this capability by adding this new condition flag. The decompressor should write back the mask only if this flag is set. This flag is used only by `ImageCodecFlush` (II-708).

`codecConditionToBuffer`

Set to 1 if the current decompression operation is decompressing into an offscreen buffer.

callerFlags Constants`codecFlagUpdatePrevious`

Controls whether your compressor updates the previous image during compression. This flag is only used with sequences that are being temporally compressed. If this flag is set to 1, your compressor should copy the current

CodecDecompressParams

frame into the previous frame buffer at the end of the frame-compression sequence. Use the source image.

`codecFlagWasCompressed`

Indicates to your compressor that the image to be compressed has been compressed before. This information may be useful to compressors that can compensate for the image degradation that may otherwise result from repeated compression and decompression of the same image. This flag is set to 1 to indicate that the image was previously compressed. This flag is set to 0 if the image was not previously compressed.

`codecFlagUpdatePreviousComp`

Controls whether your compressor updates the previous image buffer with the compressed image. This flag is only used with temporal compression. If this flag is set to 1, your compressor should update the previous frame buffer at the end of the frame compression sequence, allowing your compressor to perform frame differencing against the compression results. Use the image that results from the compression operation. If this flag is set to 0, your compressor should not modify the previous frame buffer during compression.

`codecFlagLiveGrab`

Indicates whether the current sequence results from grabbing live video. When working with live video, your compressor should operate as quickly as possible and disable any additional processing, such as compensation for previously compressed data. This flag is set to 1 when you are compressing from a live video source.

accuracy Constants

`codecMinQuality`

The minimum valid value for a CodecQ field.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a CodecQ field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

matrixFlags Constants

`matrixFlagScale2x`

Double-scale; value is $1L \ll 7$.

`matrixFlagScale1x`

Single-scale; value is $1L \ll 6$.

`matrixFlagScaleHalf`

Half-scale; value is $1L \ll 5$.

screenFloodMethod Constants

`kScreenFloodMethodNone`

No method; value is 0.

`kScreenFloodMethodKeyColor`

Key color method; value is 1.

`kScreenFloodMethodAlpha`

Alpha channel method; value is 2.

Discussion

The Image Compression Manager creates the decompression parameters structure, and your image decompressor component is required only to provide values for the `wantedDestinationPixelSize` and `wantedDestinationPixelTypes` fields of the structure. Your image decompressor component can also modify other fields if necessary. For example, if it can scale images, it must set the `codecCapabilityCanScale` flag in the `capabilities` field of the structure.

Version Notes

Some of the fields in `CodecDecompressParams` were added for various versions of QuickTime starting with version 2.1. See comments in the C interface file for details.

Associated Functions

`ImageCodecBandDecompress` (II-689)

Programming Info

C interface file: `ImageCodec.h`

See Also

See `ImageCodecBandDecompress` (II-689).

CodecInfo

CodecInfo

Describes the capabilities of a compressor.

```
struct CodecInfo {
    Str31          typeName;
    short          version;
    short          revisionLevel;
    long           vendor;
    long           decompressFlags;
    long           compressFlags;
    long           formatFlags;
    UInt8          compressionAccuracy;
    UInt8          decompressionAccuracy;
    unsigned short compressionSpeed;
    unsigned short decompressionSpeed;
    UInt8          compressionLevel;
    UInt8          resvd;
    short          minimumHeight;
    short          minimumWidth;
    short          decompressPipelineLatency;
    short          compressPipelineLatency;
    long           privateData;
};
```

typeName

Indicates the compression algorithm used by the component; for example, 'Animation'. This Pascal string may be used to identify the compression algorithm to the user. The string always takes up 32 bytes no matter how long it is. The 32 bytes consist of 31 bytes plus one length byte. Apple assigns these type names. The value of this field should correspond to the value of the typeName field in the appropriate compressor name structure returned by GetCodecNameList (I-386).

version

Indicates the version of compressed data this component supports. The contents of this field should indicate the most recent version of the compression algorithm that the component can understand.

revisionLevel

Indicates the version of the component; for example, 0x00010001 (1.0.1). Developers of compressors assign these version numbers.

vendor

Identifies the developer of the component; for example, 'appl'. The value of this field corresponds to the manufacturer code or application signature assigned to the developer.

decompressFlags

Contains flags (see below) that specify the decompression capabilities of the component. Typically, these flags are of interest only to developers of image decompressors.

compressFlags

Contains flags (see below) that specify the compression capabilities of the component. Typically, these flags are of interest only to developers of image compressors.

formatFlags

Contains flags (see below) that describe the possible format for compressed data produced by this component and the format of compressed files that the component can handle during decompression. Typically, these flags are of interest only to developers of compressor components.

compressionAccuracy

Indicates the relative accuracy of the compression algorithm employed by the component. Valid values for this field range from 0 to 255. A value of 0 means that the accuracy is unknown. Values from 1 to 255 provide a gauge for the relative accuracy of the compression algorithm; higher values indicate better accuracy. The Image Compression Manager examines this field to determine which compressor component can most accurately compress a given image. The `compressionAccuracy` field can only approximate the accuracy of a compression algorithm. Typically, compression algorithms produce results of varying quality based on a variety of parameters, including image size and content. Since this information is not available until a compression request is issued, a precise measure of accuracy is not possible. However, the value of this field should still give a rough idea of the accuracy of the supported algorithm.

decompressionAccuracy

Indicates the relative accuracy of the decompression algorithm employed by the component. Valid values for this field range from 0 to 255. A value of 0 means that the accuracy is unknown. Values from 1 to 255 indicate the relative accuracy of the decompression technique; higher values mean better accuracy. The Image Compression Manager examines this field to determine which decompressor component can most accurately decompress a given image. The `decompressionAccuracy` field can only approximate the accuracy of a decompression algorithm. Typically, decompression algorithms produce

results of varying quality based on a variety of parameters, including image size and content. Since this information is not available until a decompression request is issued, a precise measure of accuracy is not possible. However, the value of this field should still give a rough idea of the accuracy of the supported algorithm.

`compressionSpeed`

Indicates the relative speed of the component for compression operations. Valid values for this field lie in the range from 0 to 65,535. A value of 0 means that the speed is unknown. Values from 1 to 65,535 correspond to the number of milliseconds the component requires to compress a 320-by-240 pixel image on a Macintosh II computer. The Image Compression Manager examines this field to determine which compressor component can most quickly compress a given image.

`decompressionSpeed`

Indicates the relative speed of the component for decompression operations. Valid values for this field lie in the range from 0 to 65,535. A value of 0 means that the speed is unknown. Values from 1 to 65,535 correspond to the number of milliseconds the component requires to decompress a 320-by-240 pixel image on a Macintosh II computer. The Image Compression Manager examines this field to determine which compressor component can most quickly decompress a given image.

`compressionLevel`

Indicates the relative compression achieved by this component. Valid values for this field lie in the range from 0 to 255. A value of 0 means that the compression level is unknown. Values from 1 to 255 map to percentage values of relative compression; lower values mean lesser compression. A value of 1 means no compression (0 percent); a value of 255 means maximum compression (100 percent). The Image Compression Manager examines this field to determine which available compressor component will yield the smallest resulting data for a given image. The `compressionLevel` field can only approximate the effectiveness of a compression algorithm. Typically, compression algorithms produce results of varying quality based on a variety of parameters, including image size and content. Since this information is not available until a compression request is issued, a precise measure of compression is not possible. However, the value of this field should still give a rough idea of the effectiveness of the supported algorithm.

`resvd`

Reserved; set to 0.

`minimumHeight`

Specifies the height in pixels of the smallest image the component can handle. Together with the `minimumWidth` field, this field defines the block size for the component. The Image Compression Manager does not issue compression or decompression requests for images smaller than the block size.

`minimumWidth`

Specifies the width in pixels of the smallest image the component can handle. Together with the `minimumHeight` field, this field defines the block size for the component. The Image Compression Manager does not issue compression or decompression requests for images smaller than the block size.

`decompressPipelineLatency`

Decompression pipeline latency in milliseconds, for asynchronous codecs.

`compressPipelineLatency`

Compression pipeline latency in milliseconds, for asynchronous codecs.

`privateData`

Reserved for future use. This field must be set to 0.

decompressFlags and compressFlags Constants

`codecInfoDoes1`

Codec can work with 1-bit pixels.

`codecInfoDoes2`

Codec can work with 2-bit pixels.

`codecInfoDoes4`

Codec can work with 4-bit pixels.

`codecInfoDoes8`

Codec can work with 8-bit pixels.

`codecInfoDoes16`

Codec can work with 16-bit pixels.

`codecInfoDoes32`

Codec can work with 32-bit pixels.

`codecInfoDoesDither`

Codec can **dither** images.

`codecInfoDoesStretch`

Codec can stretch images to arbitrary sizes.

`codecInfoDoesShrink`

Codec can shrink images to arbitrary sizes.

CodecInfo

`codecInfoDoesMask`

Codec can mask images to clipping regions.

`codecInfoDoesTemporal`

Codec can handle temporal redundancy.

`codecInfoDoesDouble`

Codec can stretch images to exactly double size.

`codecInfoDoesQuad`

Codec can stretch images to exactly quadruple size.

`codecInfoDoesHalf`

Codec can shrink images to exactly half size.

`codecInfoDoesQuarter`

Codec can shrink images to exactly quarter size.

`codecInfoDoesRotate`

Codec can rotate images during decompression.

`codecInfoDoesHorizFlip`

Codec can flip images horizontally during decompression.

`codecInfoDoesVertFlip`

Codec can flip images vertically during decompression.

`codecInfoHasEffectParameterList`

Codec implements `QTGetEffectsList` (II-1174).

`codecInfoDoesBlend`

Codec can blend image during decompression.

`codecInfoDoesWarp`

Codec can warp image arbitrarily during decompression.

`codecInfoDoesRecompress`

Codec can recompress image without accumulating errors.

`codecInfoDoesSpool`

Codec can spool image data.

`codecInfoDoesRateConstrain`

Codec can constrain data rate.

formatFlags Constants

`codecInfoDepth1`

Compressed data available at 1 bit-per-pixel depth.

`codecInfoDepth2`

Compressed data available at 2 bit-per-pixel depth.

`codecInfoDepth4`

Compressed data available at 4 bit-per-pixel depth.

`codecInfoDepth8`

Compressed data available at 8 bit-per-pixel depth.

`codecInfoDepth16`

Compressed data available at 16 bit-per-pixel depth.

`codecInfoDepth24`

Compressed data available at 24 bit-per-pixel depth.

`codecInfoDepth32`

Compressed data available at 32 bit-per-pixel depth.

`codecInfoDepth33`

Compressed data available at 1 bit-per-pixel monochrome depth.

`codecInfoDepth34`

Compressed data available at 2 bit-per-pixel grayscale depth.

`codecInfoDepth36`

Compressed data available at 4 bit-per-pixel grayscale depth.

`codecInfoDepth40`

Compressed data available at 8 bit-per-pixel grayscale depth.

`codecInfoStoresClut`

Compressed data can have custom **color lookup tables**.

`codecInfoDoesLossless`

Compressed data can be stored in lossless format.

`codecInfoSequenceSensitive`

Compressed data is sensitive to out-of-sequence decoding.

Discussion

Contains the description of a codec.

Version Notes

The `codecInfoHasEffectParameterList` constant was formerly `codecInfoDoesSkew`.

Associated Functions

`GetCodecInfo` (I–385)

Programming Info

C interface file: `ImageCompression.h`

See Also

See the `CodecNameSpec` (IV–2208) structure and the `GetCodecNameList` (I–386) function.

CodecNameSpec

CodecNameSpec

Defines a compressor name.

```
struct CodecNameSpec {  
    CodecComponent    codec;  
    CodecType         cType;  
    Str31             typeName;  
    Handle             name;  
};
```

codec

Uniquely identifies the component or, in some cases, contains a special value that selects all components. If your application requests a list of components, the `codec` field in each compressor name structure contains the component ID for that compressor. If your application requests a list of component types, the `codec` field is set to 0 in each compressor name structure

cType

Contains the type identifier for the compressor. The value of this field indicates the compression algorithm supported by the component. See “Codec Identifiers” (IV–2655) for a list of values.

typeName

Contains a text string in Pascal format that identifies the compression algorithm supported by the component. This string may be used to identify the compression algorithm to the user. The value of this field should correspond to the value of the `typeName` field in the appropriate compressor information structure returned by the component in response to `GetCodecInfo` (I–385).

name

Specifies the name of the compressor component. You can assign this name to uniquely identify your product, and it may be used to identify the component to the user.

Programming Info

C interface file: `ImageCompression.h`

See Also

See the `CodecInfo` (IV–2202) structure and the `GetCodecInfo` (I–385) function.

CodecNameSpecList

Contains a list of `CodecNameSpec` (IV–2208) structures.

```
struct CodecNameSpecList {
    short      count;
    CodecNameSpec list[1];
};
```

count

Indicates the number of compressor name structures contained in the list array that follows.

list

Contains an array of compressor name structures. Each structure corresponds to one compressor component or type that meets the selection criteria your application specifies when it calls `GetCodecNameList` (I–386).

Associated Functions

`DisposeCodecNameList` (I–257)

Programming Info

C interface file: `ImageCompression.h`

See Also

The `GetCodecNameList` (I–386) function returns a `CodecNameSpecList` structure.

ColorSpec

Associates an RGB color value with an index or other value.

```
struct ColorSpec {
    short      value;
    RGBColor   rgb;
};
```

value

The pixel value assigned by Color QuickDraw for the color specified in the `rgb` field of this structure. Color QuickDraw assigns a pixel value based on the capabilities of the user's screen. For indexed devices, the pixel value is an index number assigned by QuickTime to the closest color available on the indexed device; for direct devices, this value expresses the best available red, green, and blue values for the color on the direct device.

rgb

An `RGBColor` (IV–2403) structure that fully specifies the color whose approximation Color QuickDraw specifies in the `value` field.

ColorTable

Discussion

When creating a `PixMap` (IV–2332) structure for an indexed device, `Color QuickDraw` creates a `ColorTable` (IV–2210) structure that defines the best colors available for the pixel image on that graphics device. The Color Manager also stores a `ColorTable` structure for the currently available colors in the graphics device’s **color lookup table**.

Programming Info

C interface file: `Quickdraw.h`

See Also

See `ColorTable` (IV–2210).

ColorTable

Provides a color table for indexed color spaces.

```
struct ColorTable {
    long          ctSeed;
    short         ctFlags;
    short         ctSize;
    CSpecArray    ctTable;
};
```

ctSeed

Identifies a particular instance of a color table. The Color Manager uses the `ctSeed` value to compare an indexed device’s color table with its associated inverse table (a table it uses for fast color lookup). When the color table for a graphics device has been changed, the Color Manager needs to rebuild the inverse table.

ctFlags

A flag that distinguishes pixel map color tables from color tables in `GDevice` records. If the high bit is 0, this is a pixel map color table; if the high bit is 1, this is a color tables in a `GDevice` (IV–2264) structure.

ctSize

One less than the number of entries in the table.

ctTable

An array of `ColorSpec` (IV–2209) structures, each containing a pixel value and a color specified by an `RGBColor` (IV–2403) structure.

Discussion

When creating a `Pixmap` (IV–2332) structure for an indexed device, `Color QuickDraw` creates a `ColorTable` (IV–2210) structure that defines the best colors available for the pixel image on that graphics device. The `Color Manager` also stores a `ColorTable` structure for the currently available colors in the graphics device's **color lookup table**. Typically, your application needs to create `ColorTable` records only if it uses the `Mac OS Palette Manager`.

Programming Info

C interface file: `Quickdraw.h`

See Also

See `ColorSpec` (IV–2209). For the inverse structure, see `ITab` (IV–2297).

ComponentAliasResource

Provides an **alias** to a component that can handle more than one data type; for example, a WAV importer can be an alias to the AIFF importer code, which reads WAV files.

```
struct ComponentAliasResource {
    ComponentResource      cr;
    ComponentDescription    aliasCD;
};
```

`cr`

Information needed to register the component. This information is used in the same way as registration information for ordinary components, with one exception: the specification of the code **resource** for the component is ignored.

`aliasCD`

The target for the **alias**. The code **resource** of the target is used in place of the code resource specified in the `cr` field.

Discussion

To resolve a component **alias**, the `Component Manager` passes the contents of the `aliasCD` field to `FindNextComponent` (I–360), just as an application would. This field includes all the information that is necessary to specify the target component. If there is no component that matches the specifications, the request to open the alias fails. The `Component Manager` never resolves a component alias until the alias is opened. This lets you register component aliases before their target components are registered. It also lets you replace target components with other components.

ComponentDescription

Programming Info

C interface file: `Components.h`

ComponentDescription

Describes a class of components by their attributes.

```
struct ComponentDescription {
    OSType          componentType;
    OSType          componentSubType;
    OSType          componentManufacturer;
    unsigned long   componentFlags;
    unsigned long   componentFlagsMask;
};
```

componentType

A four-character code that identifies the type of component. See “Codec Identifiers” (IV–2655). All components of a particular type support a common set of interface routines. Your application uses this field to search for components of a given type.

componentSubType

A four-character code that identifies the subtype of the component. Different subtypes of a component type may support additional features or provide interfaces that extend beyond the standard routines for a given component type. For example, the subtype of an image compressor component indicates the compression algorithm employed by the compressor. Your application can use the `componentSubType` field to perform a more specific lookup operation than is possible using only the `componentType` field. Set this parameter to 0 to select a component with any subtype value.

componentManufacturer

A four-character code that identifies the manufacturer of the component. This field allows for further differentiation between individual components. For example, components made by a specific manufacturer may support an extended feature set. Components provided by Apple have a manufacturer value of 'appl'. Your application can use this field to find components from a certain manufacturer. A value of 0 operates as a wildcard

componentFlags

A 32-bit field that contains flags describing your component’s capabilities. The high-order 8 bits are defined by the Component Manager. You should set these bits to 0. The low-order 24 bits are specific to each component type. These flags

can be used to indicate the presence of features or capabilities in a given component (see below).

`componentFlagsMask`

A 32-bit field that indicates which flags in the `componentFlags` field are relevant to a particular component search operation. For each flag in the `componentFlags` field that is to be considered as a search criterion, your application should set the corresponding bit in this field to 1. To ignore a flag, set the bit to 0.

componentFlags Constants

`canMovieExportAuxDataHandle`

Set this bit if your export component exports to an auxiliary data handle. A movie export component that supports `MovieExportGetAuxiliaryData` (II-923) should set this flag in its `componentFlags` parameter.

`canMovieImportValidateHandles`

Set this bit if your movie import component can and wants to validate handles.

`canMovieImportValidateFile`

Set this bit if your movie import component can and wants to validate files.

`dontRegisterWithEasyOpen`

Set this bit when Macintosh Easy Open is installed and you do not want QuickTime to register your component with Easy Open (preferring to handle interactions with Macintosh Easy Open yourself).

`canMovieImportInPlace`

Set this bit if your movie import component can create a movie from a file without having to write to a separate disk file. Examples include MPEG and AIFF import components.

`movieImportSubTypeIsFileExtension`

Set this bit if your component's subtype is a file extension instead of a Macintosh file type. For example, if your import component opens files with an extension of `.doc`, you would set this flag and set your component subtype to `'DOC '`.

`canMovieImportPartial`

Set this bit if your movie import component implements `MovieImportSetOffsetAndLimit` (II-959).

`hasMovieImportMIMEList`

Set this bit if you movie import component supports returning a MIME type list. A movie import component that supports `MovieImportGetMIMETypeList` (II-948) should set this flag in the `componentFlags` field of the component description record.

ComponentDescription

`canMovieExportFromProcedures`

Set this bit if your movie export component supports `MovieExportFromProceduresToDataRef` (II-922).

`canMovieExportValidateMovie`

Set this bit if your movie export component validates the movie passed to it.

`movieExportNeedsResourceFork`

Set this bit if your export component requires the creation of the **resource** fork of a Mac OS file.

`canMovieImportDataReferences`

Set this bit if your movie import component can accept a data reference (such as a handle) as the source for the import.

`movieExportMustGetSourceMediaType`

Set this bit if the export component implements the new exporter component registration mechanism. The component must implement `MovieExportGetSourceMediaType` (II-927).

`canMovieImportValidateDataReferences`

Set this bit if the export component supports `MovieImportValidateDataRef` (II-965).

`canMovieImportHandles`

Set this bit if your movie import component can import from a handle.

`canMovieImportFiles`

Set this bit if your movie import component can import files.

`hasMovieImportUserInterface`

Set this bit if your movie import component has its own user interface.

`canMovieExportHandles`

Set this bit if your movie export component can export to a handle.

`canMovieExportFiles`

Set this bit if your movie export component can export to a file.

`hasMovieExportUserInterface`

Set this bit if your movie export component has its own user interface.

`dontAutoFileMovieImport`

Set this bit to turn off automatic file conversion.

`channelFlagDontOpenResFile`

Instructs the sequence grabber not to open your panel component's **resource** file. By default, the sequence grabber opens your component's resource file for you, and then provides you with the appropriate file reference number. In general, this is convenient. However, if your component is linked with your

application and does not have its own resource file, you may not want the sequence grabber to try to open the resource file. In such cases, set this flag to 1. For an example, see `SGPanelGetDit1` (III-1761).

`channelFlagHasDependency`

Lets you tell the sequence grabber that your panel component requires special digitizing hardware. If you set this flag to 1, the sequence grabber gives your component an opportunity to verify that it can work in the current hardware environment, by calling your component's `SGPanelCanRun` (III-1759) function.

Discussion

The Component Manager ignores fields in the component description structure that are set to 0. For example, if you set all the fields to 0 and pass this structure to `CountComponents` (I-143), the Component Manager counts the total number of components registered in the system. If you set all fields to 0 except for the `componentManufacturer` field, the Component Manager counts the number of registered components supplied by the manufacturer you specify.

Associated Functions

`CountComponents` (I-143)

Programming Info

C interface file: `Components.h`

ComponentInstanceRecord

Undocumented

```
struct ComponentInstanceRecord {
    long    data[1];
};
```

`data`

Undocumented

Discussion

Undocumented

Programming Info

C interface file: `Components.h`

ComponentMPWorkFunctionHeaderRecord

Stores data for the `ComponentMPWorkFunctionProc` (III-2077) callback.

ComponentParameters

```
struct ComponentMPWorkFunctionHeaderRecord {
    UInt32    headerSize;
    UInt32    recordSize;
    UInt32    workFlags;
    UInt16    processorCount;
    UInt8     unused;
    UInt8     isRunning;
};
```

headerSize
Undocumented

recordSize
Undocumented

workFlags
Undocumented

processorCount
Undocumented

unused
Undocumented

isRunning
Undocumented

Discussion

Undocumented

Associated Functions

ComponentMPWorkFunctionProc (III-2077)

Programming Info

C interface file: Components.h

ComponentParameters

Data structure passed by the Component Manager to a component.

```
struct ComponentParameters {
    UInt8    flags;
    UInt8    paramSize;
    short    what;
    long     params[1];
};
```

flags

Reserved.

paramSize

Size in bytes of actual parameters passed.

what

The type of request (see below). Negative values are reserved for definition by Apple. You can use values greater than or equal to 0 to define other requests that are supported by your component.

params

Parameters sent to your component.

what Constants

kComponentOpenSelect

Open a connection; required.

kComponentCloseSelect

Close an open connection; required.

kComponentCanDoSelect

Determine whether your component supports a particular request; required.

kComponentVersionSelect

Return your component's version number; required.

kComponentRegisterSelect

Determine whether your component can operate in the current environment.

kComponentTargetSelect

Call another component whenever it would call itself (as a result of your component being used by another component).

kComponentUnregisterSelect

Perform any operations that are necessary as a result of your component being unregistered.

kComponentGetMPWorkFunctionSelect

Undocumented

kComponentExecuteWiredActionSelect

Undocumented

kComponentGetPublicResourceSelect

Undocumented

Discussion

Whenever the Component Manager receives a request for your component, it calls your component's entry point and passes any parameters, along with information

ComponentPlatformInfo

about the current connection, in this structure. The Component Manager also passes a handle to the global storage (if any) associated with that instance of your component. When your component receives a request, it should examine the parameters to determine the nature of the request, perform the appropriate processing, set an error code if necessary, and return an appropriate function result to the Component Manager.

Associated Functions

CallComponent (I-58)

Programming Info

C interface file: Components.h

ComponentPlatformInfo

Indicates the platform a component is ready to run on.

```
struct ComponentPlatformInfo {  
    long          componentFlags;  
    ResourceSpec  component;  
    short         platformType;  
};
```

componentFlags

Component flags (different for each component type).

component

The **resource** where the component code is found.

platformType

Platform code (see below).

platformType Constants

platform68k

Motorola MC68k architecture.

platformPowerPC

IBM PowerPC architecture.

platformInterpreted

Undocumented

platformWin32

Microsoft Windows 32 architecture.

platformPowerPCNativeEntryPoint

Native PowerPC architecture.

Programming Info

C interface file: Components.h

ComponentPlatformInfoArray

Contains an array of ComponentPlatformInfo (IV-2218) structures.

```
struct ComponentPlatformInfoArray {
    long          count;
    ComponentPlatformInfo platformArray[1];
};
```

count

Number of ComponentPlatformInfo (IV-2218) structures in the array.

platformArray

The array.

Programming Info

C interface file: Components.h

ComponentRecord

Undocumented

```
struct ComponentRecord {
    long    data[1];
};
```

data

Undocumented

Discussion

Undocumented

Programming Info

C interface file: Components.h

ComponentResource

Defines the structure of a component **resource**.

```
struct ComponentResource {
```

ComponentResource

```
    ComponentDescription    cd;  
    ResourceSpec            component;  
    ResourceSpec            componentName;  
    ResourceSpec            componentInfo;  
    ResourceSpec            componentIcon;  
};
```

cd

A `ComponentDescription` (IV–2212) structure that specifies the characteristics of the component.

component

A `ResourceSpec` (IV–2402) structure that specifies the type and ID of the component code **resource**. The `resType` field of this structure may contain any value. The component’s main entry point must be at offset 0 in the resource.

componentName

A `ResourceSpec` (IV–2402) structure that specifies the **resource** type and ID for the name of the component. This is a Pascal string. Typically, the component name is stored in a resource of type 'STR '.

componentInfo

A `ResourceSpec` (IV–2402) structure that specifies the **resource** type and ID for the information string that describes the component. This is a Pascal string. Typically, the information string is stored in a resource of type 'STR '. You might use the information stored in this resource in a Get Info dialog box.

componentIcon

A `ResourceSpec` (IV–2402) structure that specifies the **resource** type and ID for the icon for a component. Component icons are stored as 32-by-32 bit maps. Typically, the icon is stored in a resource of type 'ICON'. Note that this icon is not used by the Finder; you supply an icon only so that other components or applications can display your component’s icon in a dialog box if needed.

Discussion

You can optionally append to the end of this structure the information defined by `ComponentResourceExtension` (IV–2221).

Associated Functions

`RegisterComponentResource` (III–1453)

Programming Info

C interface file: `Components.h`

See Also

See `ExtComponentResource` (IV–2249).

ComponentResourceExtension

Extends the ComponentResource (IV–2219) structure.

```
struct ComponentResourceExtension {
    long    componentVersion;
    long    componentRegisterFlags;
    short   componentIconFamily;
};
```

`componentVersion`

The version number of the component. If you specify the `componentDoAutoVersion` flag in `componentRegisterFlags`, the Component Manager must obtain the version number of your component when your component is registered. Either you can provide a version number in your component's **resource**, or you can specify a value of 0 for its version number. If you specify 0, the Component Manager sends your component a version request to get the version number of your component.

`componentRegisterFlags`

A set of flags (see below) containing additional registration information.

`componentIconFamily`

The **resource** ID of an icon family. You can provide an icon family in addition to the icon provided in the `componentIcon` field of ComponentResource (IV–2219). Note that members of this icon family are not used by the Finder; you supply an icon family only so that other components or applications can display your component's icon in a dialog box if needed.

componentRegisterFlags Constants

`componentDoAutoVersion`

Tells the Component Manager to resolve conflicts between different versions of the same component. If you specify this flag, the Component Manager registers your component only if there is no later version available. If an older version is already registered, the Component Manager unregisters it. If a newer version of the same component is registered after yours, the Component Manager automatically unregisters your component. You can use this automatic version control feature to make sure that the most recent version of your component is registered, regardless of the number of versions that are installed.

`componentWantsUnregister`

You want your component to receive an unregister request when it is unregistered.

CompressionInfo

componentAutoVersionIncludeFlags

You want the Component Manager to include the `componentFlags` field of the `ComponentDescription` (IV–2212) structure when it searches for identical components in the process of performing automatic version control for your component. If you do not specify this flag, the Component Manager searches only the `componentType`, `componentSubType`, and `componentManufacturer` fields. When the Component Manager performs automatic version control for your component, it searches for components with identical values in the `componentType`, `componentSubType`, and `componentManufacturer` fields (and optionally, in the `componentFlags` field). If it finds a matching component, it compares version numbers and registers the most recent version of the component. Note that the setting of this flag affects automatic version control only and does not affect the search operations performed by `FindNextComponent` and `CountComponents`.

componentHasMultiplePlatforms

Undocumented

componentLoadResident

Undocumented

Programming Info

C interface file: `Components.h`

CompressionInfo

Returns sound compression information to the Sound Manager.

```
struct CompressionInfo {
    long          recordSize;
    OSType        format;
    short         compressionID;
    unsigned short samplesPerPacket;
    unsigned short bytesPerPacket;
    unsigned short bytesPerFrame;
    unsigned short bytesPerSample;
    unsigned short futureUse1;
};
```

`recordSize`

The size in bytes of this structure.

`format`

The compression format.

compressionID
The compression ID.

samplesPerPacket
The number of samples in each packet.

bytesPerPacket
The number of bytes in each packet.

bytesPerFrame
The number of bytes in each frame.

bytesPerSample
The number of bytes in each sample.

futureUse1
Reserved. Set this field to 0.

Discussion

When the Sound Manager calls `SoundComponentGetInfo` (III–1849) with the `siCompressionFactor` selector, you need to return a pointer to this structure.

Associated Functions

`GetCompressionInfo` (I–398)

Programming Info

C interface file: `Sound.h`

ConnectionSpeedPrefsRecord

The record of connection speed.

```
struct ConnectionSpeedPrefsRecord {
    long    connectionSpeed;
};
```

connectionSpeed
Connection speed.

Discussion

The following code shows a typical use of `ConnectionSpeedPrefsRecord`.

```
OSErr err;
QTAtomContainer prefs;
QTAtom prefsAtom;
long dataSize;
Ptr atomData;
```

ControlBehaviors

```
ConnectionSpeedPrefsRecord prefrec;

err = GetQuickTimePreference(ConnectionSpeedPrefsType, &prefs);
if (err == noErr) {
    prefsAtom = QTFindChildByID(prefs, kParentAtomIsContainer,
                                ConnectionSpeedPrefsType, 1, NIL);

    if (!prefsAtom) {
        // set the default setting to 28.8kpbs
        prefrec.connectionSpeed = kDataRate288ModemRate;
    } else {
        err = QTGetAtomDataPtr(prefs, prefsAtom, &dataSize, &atomData);
        if (dataSize != sizeof(ConnectionSpeedPrefsRecord)) {
            // the prefs record wasn't the right size,
            // so it must be corrupt -- set to the default
            prefrec.connectionSpeed = kDataRate288ModemRate;
        } else {
            // everything was fine -- read the connection speed
            prefrec = *(ConnectionSpeedPrefsRecord *)atomData;
        }
    }
    QTDisposeAtomContainer(prefs);
}
```

Programming Info

C interface file: Movies.h

ControlBehaviors

Provides data for the ParameterDataBehavior (IV–2328) structure.

```
struct ControlBehaviors {
    QTAtomID    groupID;
    long        controlValue;
};
```

groupID

Group under control of this item.

controlValue

Control value for comparison purposes.

Programming Info

C interface file: ImageCodec.h

See Also

See `ParameterDataBehavior` (IV–2328).

ConversionBlock

Undocumented

```
struct ConversionBlock {
    short          destination;
    short          unused;
    CmpSoundHeaderPtr inputPtr;
    CmpSoundHeaderPtr outputPtr;
};
```

`destination`

Undocumented

`unused`

Undocumented

`inputPtr`

Undocumented

`outputPtr`

Undocumented

Programming Info

C interface file: `Sound.h`

CProcRec

Provides data for a `GDevice` (IV–2264) structure.

```
struct CProcRec {
    Handle          nxtComp;
    ColorComplementUPP compProc;
};
```

`nxtComp`

Handle to next `CProcRec` structure.

CQDProcs

compProc

A **Universal Procedure Pointer** that accesses a `ColorComplementProc` callback; see the chapter “Color Manager” in *Advanced Color Imaging on the Mac OS* (listed in the **bibliography**).

Programming Info

C interface file: `Quickdraw.h`

See Also

See the `gdCompProc` field of the `GDevice` (IV–2264) structure.

CQDProcs

Accesses QuickDraw’s low-level routines.

```
struct CQDProcs {
    QDTextUPP          textProc;
    QDLineUPP          lineProc;
    QDRectUPP          rectProc;
    QDRRectUPP         rRectProc;
    QDOvalUPP          ovalProc;
    QDArcUPP           arcProc;
    QDPolyUPP          polyProc;
    QDRgnUPP           rgnProc;
    QDBitsUPP          bitsProc;
    QDCommentUPP       commentProc;
    QDTxMeasUPP        txMeasProc;
    QDGetPicUPP        getPicProc;
    QDPutPicUPP        putPicProc;
    QDOpcodeUPP        opcodeProc;
    UniversalProcPtr   newProc1;
    QDStdGlyphsUPP     glyphsProc;
    QDPrinterStatusUPP printerStatusProc;
    UniversalProcPtr   newProc4;
    UniversalProcPtr   newProc5;
    UniversalProcPtr   newProc6;
};
```

textProc

A pointer to the low-level routine that draws text. The standard QuickDraw routine is the `StdText` procedure.

lineProc

A pointer to the low-level routine that draws lines. The standard QuickDraw routine is the `StdLine` procedure.

`rectProc`

A pointer to the low-level routine that draws rectangles. The standard QuickDraw routine is the `StdRect` procedure.

`rRectProc`

A pointer to the low-level routine that draws rounded rectangles. The standard QuickDraw routine is the `StdRRect` procedure.

`ovalProc`

A pointer to the low-level routine that draws ovals. The standard QuickDraw routine is the `StdOval` procedure.

`arcProc`

A pointer to the low-level routine that draws arcs. The standard QuickDraw routine is the `StdArc` procedure.

`polyProc`

A pointer to the low-level routine that draws polygons. The standard QuickDraw routine is the `StdPoly` procedure.

`rgnProc`

A pointer to the low-level routine that draws regions. The standard QuickDraw routine is the `StdRgn` procedure.

`bitsProc`

A pointer to the low-level routine that copies bitmaps. The standard QuickDraw routine is the `StdBits` procedure.

`commentProc`

A pointer to the low-level routine for processing a picture comment. The standard QuickDraw routine is the `StdComment` procedure.

`txMeasProc`

A pointer to the low-level routine for measuring text width. The standard QuickDraw routine is the `StdTxMeas` function.

`getPicProc`

A pointer to the low-level routine for retrieving information from the definition of a picture. The standard QuickDraw routine is the `StdGetPic` procedure.

`putPicProc`

A pointer to the low-level routine for saving information as the definition of a picture. The standard QuickDraw routine is the `StdPutPic` procedure.

`opcodeProc`

A **Universal Procedure Pointer** that accesses a `QD0opcodeProc` (III–2116) callback.

`newProc1`

The `StdPix` bottleneck; see `ImageCompression.h`.

DataHChokeAtomRecord

glyphsProc

A **Universal Procedure Pointer** that accesses a QDStdGlyphsProc (III–2122) callback. This callback is used for **Unicode** text drawing.

newProc2

Procedure used in **Unicode** text drawing.

printerStatusProc

Procedure used to communicate status between printing code and system imaging code.

newProc3 newProc4 newProc5 newProc6

Procedures used to communicate status between printing code and system imaging code.

Programming Info

C interface file: Quickdraw.h

DataHChokeAtomRecord

Undocumented

```
struct DataHChokeAtomRecord {  
    long    flags;  
    long    param;  
};
```

flags

A constant (see below) that defines the metrics of the param parameter.

param

Choke rate (see below)

flags Constants

kDataHChokeToMovieDataRate

The param parameter is 0.

kDataHChokeToParam

The param parameter is bytes per second.

Programming Info

C interface file: QuickTimeComponents.h

DataHScheduleRecord

Provides scheduling information for scheduled reads.

```
struct DataHScheduleRecord {
    TimeRecord    timeNeededBy;
    long          extendedID;
    long          extendedVers;
    Fixed         priority;
};
```

timeNeededBy

Specifies the time at which your data handler must deliver the requested data to the calling program. This time value is relative to the time base that is contained in this time record. During pre-roll operations, the Movie Toolbox may use special values in certain time record fields. The time record fields in question are the scale and value fields. By correctly interpreting the values of these fields, your data handler can queue up the pre-roll read requests in the most efficient way for its device.

extendedID

A constant (see below) that indicates the type of data that follows in the remainder of the structure.

extendedVers

Reserved. This field should always be set to 0.

priority

Indicates the relative importance of the data request. Client programs assign a value of 100.0 to data requests that must be delivered. Lower values indicate relatively less critical data. If your data handler must accommodate **bandwidth** limitations when delivering data, your component may use this value as an indication of which requests can be dropped with the least impact on the client program. As an example, consider using priorities in a frame-differenced movie. Key frames might have priority values of 100.0, indicating that they are essential to proper playback. As you move through the frames following a **key frame**, each successive frame might have a lower priority value. Once you drop a frame, you must drop all successive frames of equal or lower priority until you reach another key frame, because each of these frames would rely on the dropped one for some image data.

extendedID Constants

kDataHExtendedSchedule

The remainder of the structure contains extended scheduling information.

DataHVolumeListRecord

Discussion

There are two types of preroll read operations. The first type is a required read; that is, the Movie Toolbox requires that the read operation be satisfied before the movie starts playing. The second type is an optional read. If your data handler can satisfy the read operation as part of the pre-roll operation, it should do so. Otherwise, your data handler may satisfy the request at a specified time while the movie is playing. The Movie Toolbox indicates that a preroll read request is required by setting the `scale` field of the time record to `-1`. This literally means that the request is scheduled for a time that is infinitely far into the future. Your data handler should collect all such read requests, order them most efficiently for your device, and process them when the Movie Toolbox calls your component's `DataHFinishData` (I-182) function. For optional preroll read requests, the Movie Toolbox sets the `scale` field properly, but negates the contents of the `value` field. Your data handler has the option of delivering the data for this request with the required data, if that can be done efficiently. Otherwise, your data handler may deliver the data at its schedule time. You determine the scheduled time by negating the contents of the `value` field (that is, multiplying by `-1`).

Programming Info

C interface file: `QuickTimeComponents.h`

DataHVolumeListRecord

Contains data-handler access information for a volume.

```
struct DataHVolumeListRecord {
    short    vRefNum;
    long     flags;
};
```

`vRefNum`

The volume reference number assigned to the volume.

`flags`

Flags that indicate the level of support your data handler can provide for this volume. These flags are similar to those defined for `DataHCanUseDataRef` (I-175), though there is one additional flag. Your component should set appropriate flags to 1 and set unused flags to 0.

flags Constants

`kDataHCanRead`

Indicates that your data handler can read from the volume.

kDataHSpecialRead

Indicates that your data handler can read from the volume using a specialized method. For example, your data handler might support access to networked multimedia servers using a special protocol. In that case, your component would set this flag to 1 whenever the volume resides on a supported server.

kDataHSpecialReadFile

Reserved for use by Apple.

kDataHCanWrite

Indicates that your data handler can write data to the volume. In particular, use this flag to indicate that your data handler's `DataHPutData` (I-216) function will work with this volume.

kDataHSpecialWrite

Indicates that your data handler can write to the volume using a specialized method. As with the `kDataHSpecialRead` flag, your data handler would use this flag to indicate that your component can access the volume using specialized support (for example, special network protocols).

kDataHCanStreamingWrite

Indicates that your data handler can support the special write functions for capturing movie data when writing to this volume.

kDataHMustCheckDataRef

Instructs the calling program that your component must check each data reference before it can accurately report its capabilities. If you set this flag to 1, the Movie Toolbox will call your component's `DataHCanUseDataRef` (I-175) function before it assigns a container to your data handler. Note, however, that this may slow the data handler selection process somewhat.

Programming Info

C interface file: `QuickTimeComponents.h`

DataRateParams

Communicates information to compressors that can constrain compressed data to a specific data rate.

```
struct DataRateParams {
    long    dataRate;
    long    dataOverrun;
    long    frameDuration;
    long    keyFrameRate;
    CodecQ  minSpatialQuality;
```

DataRateParams

```
CodecQ    minTemporalQuality;  
};
```

dataRate

Specifies the bytes per second to which the data rate must be constrained.

dataOverrun

Indicates the current number of bytes above or below the desired data rate. A value of 0 means that the data rate is being met exactly. If your application doesn't know the data overrun, it should set this field to 0.

frameDuration

Specifies the duration of the current frame in milliseconds.

keyFrameRate

Indicates the frequency of **key frames**. This frequency is normally identical to the key frame rate passed to the CompressSequenceBegin (I-119).

minSpatialQuality

A constant (see below) that specifies the minimum spatial quality the compressor should use to meet the requested data rate.

minTemporalQuality

A constant (see below) that specifies the minimum temporal quality the compressor should use to meet the requested data rate.

minSpatialQuality, minTemporalQuality Constants

codecMinQuality

The minimum valid value for a CodecQ field.

codecLowQuality

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

codecNormalQuality

Image reproduction of normal quality.

codecHighQuality

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

codecMaxQuality

The maximum standard value for a CodecQ field.

codecLosslessQuality

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

The CodecQ data type defines a field that identifies the quality characteristics of a given image or sequence. Note that individual components may not implement all the quality levels shown here. In addition, components may implement other quality levels in the range from `codecMinQuality` to `codecMaxQuality`. Relative quality should scale within the defined value range. Values above `codecLosslessQuality` are reserved for use by individual components.

Associated Functions

`GetCSequenceDataRateParams` (I-403)

Programming Info

C interface file: `ImageCompression.h`

See Also

For a given compression operation, your application can determine the quality that the component supports by calling `GetCompressionTime` (I-401).

DataReferenceRecord

A data reference to data of some type.

```
struct DataReferenceRecord {
    OSType    dataRefType;
    Handle    dataRef;
};
```

`dataRefType`

Specifies the type of data reference. For an **alias** data reference, you set the parameter to `rAliasType`, indicating that the reference is an alias. For a handle data reference, set the parameter to `HandleDataHandlerSubType`.

`dataRef`

Specifies the actual data reference. This parameter contains a handle to the information that identifies the file to be used. The type of information stored in the handle depends on the value of the `dataRefType` parameter. For example, if your application is loading the movie from a file, this parameter would contain an **alias** to the movie file.

Programming Info

C interface file: `Movies.h`

DigitizerInfo

Contains information about the capabilities and current status of a video digitizer component.

```
struct DigitizerInfo {
    short      vdigType;
    long       inputCapabilityFlags;
    long       outputCapabilityFlags;
    long       inputCurrentFlags;
    long       outputCurrentFlags;
    short      slot;
    GDHandle   gdh;
    GDHandle   maskgdh;
    short      minDestHeight;
    short      minDestWidth;
    short      maxDestHeight;
    short      maxDestWidth;
    short      blendLevels;
    long       reserved;
};
```

vdigType

Constant (see below) that specifies the type of video digitizer component.

inputCapabilityFlags

Constant (see below) that specifies the capabilities of the video digitizer component with respect to the input video signal.

outputCapabilityFlags

Constant (see below) that specifies the capabilities of the video digitizer component with respect to the output digitized video information.

inputCurrentFlags

Specifies the current status of the video digitizer with respect to the input video signal. Video digitizer components report their current input status by returning a flags field that contains 1 bit for each of the applicable `inputCapabilityFlags` constants (see below), plus additional `inputCurrentFlags` constants (see below) as appropriate. The digitizer component sets these flags to reflect its current status. When reporting input status, for example, a video digitizer component sets the `digiInDoesGenLock` flag to 1 whenever the digitizer component is deriving its time signal from the input video. When reporting its input capabilities, the digitizer component sets this flag to 1 to indicate that it can derive its timing from the input video.

outputCurrentFlags

Specifies the current status of the video digitizer with respect to the output video signal. Video digitizer components report their current output status by returning a flags field that contains 1 bit for each of the applicable outputCapabilityFlags constants (see below)

slot

Identifies the slot that contains the video digitizer interface card.

gdh

Contains a handle to the graphics device that defines the screen to which the digitized data is to be written. Set this field to NIL if your application is not constrained to a particular graphics device.

maskgdh

Contains a handle to the graphics device that contains the mask plane. This field is used only by digitizers that clip by means of mask planes.

minDestHeight

Indicates the smallest height value the digitizer component can accommodate in its destination.

minDestWidth

Indicates the smallest width value the digitizer component can accommodate in its destination.

maxDestHeight

Indicates the largest height value the digitizer component can accommodate in its destination.

maxDestWidth

Indicates the largest width value the digitizer component can accommodate in its destination.

blendLevels

Specifies the number of blend levels the video digitizer component supports.

reserved

Reserved. Set this field to 0.

vdigType Constants

vdTypeBasic

Basic video digitizer; does not support any clipping.

vdTypeAlpha

Supports clipping by means of an alpha channel.

vdTypeMask

Supports clipping by means of a mask plane.

DigitizerInfo

vdTypeKey

Supports clipping by means of key colors.

inputCapabilityFlags Constants

digiInDoesNTSC

Indicates that the video digitizer supports National Television System Committee (NTSC) format input video signals. This flag is set to 1 if the digitizer component supports NTSC video.

digiInDoesPAL

Indicates that the video digitizer component supports Phase Alternation Line (PAL) format input video signals. This flag is set to 1 if the digitizer component supports PAL video.

digiInDoesSECAM

Indicates that the video digitizer component supports Systeme Electronique Couleur avec Memoire (SECAM) format input video signals. This flag is set to 1 if the digitizer component supports SECAM video.

digiInDoesGenLock

Indicates that the video digitizer component supports genlock; that is, the digitizer can derive its timing from an external time base. This flag is set to 1 if the digitizer component supports genlock.

digiInDoesComposite

Indicates that the video digitizer component supports composite input video. This flag is set to 1 if the digitizer component supports composite input.

digitInDoesSVideo

Indicates that the video digitizer component supports s-video input video. This flag is set to 1 if the digitizer component supports s-video input.

digiInDoesComponent

Indicates that the video digitizer component supports RGB input video. This flag is set to 1 if the digitizer component supports RGB input.

digiInVTR_Broadcast

Indicates that the video digitizer component can distinguish between an input signal that emanates from a videotape player and a broadcast signal. This flag is set to 1 if the digitizer component can differentiate between the two different signal types.

digiInDoesColor

Indicates that the video digitizer component supports color input. This flag is set to 1 if the digitizer component can accept color input.

digiInDoesBW

Indicates that the video digitizer component supports grayscale input. This flag is set to 1 if the digitizer component can accept grayscale input.

outputCapabilityFlags Constants

digiOutDoes1

Indicates that the video digitizer component can work with pixel maps that contain 1-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 1-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

digiOutDoes2

Indicates that the video digitizer component can work with pixel maps that contain 2-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 2-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

digiOutDoes4

Indicates that the video digitizer component can work with pixel maps that contain 4-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 4-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

digiOutDoes8

Indicates that the video digitizer component can work with pixel maps that contain 8-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 8-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

digiOutDoes16

Indicates that the video digitizer component can work with pixel maps that contain 16-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 16-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

digiOutDoes32

Indicates that the video digitizer component can work with pixel maps that contain 32-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 32-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

digiOutDoesDither

Indicates that the video digitizer component supports **dithering**. If this flag is set to 1, the component supports dithering of colors. If this flag is set to 0, the digitizer component does not support dithering.

DigitizerInfo

`digiOutDoesStretch`

Indicates that the video digitizer component can stretch images to arbitrary sizes. If this flag is set to 1, the digitizer component can stretch images. If this flag is set to 0, the digitizer component does not support stretching.

`digiOutDoesShrink`

Indicates that the video digitizer component can shrink images to arbitrary sizes. If this flag is set to 1, the digitizer component can shrink images. If this flag is set to 0, the digitizer component does not support shrinking.

`digiOutDoesMask`

Indicates that the video digitizer component can handle clipping regions. If this flag is set to 1, the digitizer component can mask to an arbitrary clipping region. If this flag is set to 0, the digitizer component does not support clipping regions.

`digiOutDoesDouble`

Indicates that the video digitizer component supports stretching to quadruple size when displaying the output video. The parameters for the stretch operation are specified in the matrix structure for the request; the component modifies the scaling attributes of the matrix. If this flag is set to 1, the digitizer component can stretch an image to exactly four times its original size, up to the maximum size specified by the `maxDestHeight` and `maxDestWidth` fields. If this flag is set to 0, the digitizer component does not support stretching to quadruple size.

`digiOutDoesQuad`

Indicates that the video digitizer component supports stretching an image to 16 times its original size when displaying the output video. The parameters for the stretch operation are specified in the matrix structure for the request; the component modifies the scaling attributes of the matrix. If this flag is set to 1, the digitizer component can stretch an image to exactly 16 times its original size, up to the maximum size specified by the `maxDestHeight` and `maxDestWidth` fields. If this flag is set to 0, the digitizer component does not support this capability.

`digiOutDoesQuarter`

Indicates that the video digitizer component can shrink an image to one-quarter of its original size when displaying the output video. The parameters for the shrink operation are specified in the matrix structure for the request; the component modifies the scaling attributes of the matrix. If this flag is set to 1, the digitizer component can shrink an image to exactly one-quarter of its original size, down to the minimum size specified by the `minDestHeight` and `minDestWidth` fields. If this flag is set to 0, the digitizer component does not support this capability.

digiOutDoesSixteenth

Indicates that the video digitizer component can shrink an image to 1/16 of its original size when displaying the output video. The parameters for the shrink operation are specified in the matrix structure for the request; the digitizer component modifies the scaling attributes of the matrix. If this flag is set to 1, the digitizer component can shrink an image to exactly 1/16 of its original size, down to the minimum size specified by the `minDestHeight` and `minDestWidth` fields. If this flag is set to 0, the digitizer component does not support this capability.

digiOutDoesRotate

Indicates that the video digitizer component can rotate an image when displaying the output video. The parameters for the rotation are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can rotate the image. If this flag is set to 0, the digitizer component cannot rotate the resulting image.

digiOutDoesHorizFlip

Indicates that the video digitizer component can flip an image horizontally when displaying the output video. The parameters for the horizontal flip are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can flip the image. If this flag is set to 0, the digitizer component cannot flip the resulting image.

digiOutDoesVertFlip

Indicates that the video digitizer component can flip an image vertically when displaying the output video. The parameters for the vertical flip are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can flip the image. If this flag is set to 0, the digitizer component cannot flip the resulting image.

digiOutDoesSkew

Indicates that the video digitizer component can skew an image when displaying the output video. Skewing an image distorts it linearly along only a single axis; for example, drawing a rectangular image into a parallelogram-shaped region. The parameters for the skew operation are specified in the matrix structure for the request. If this flag is set to 1, the digitizer component can skew an image. If this flag is set to 0, the digitizer component does not support this capability.

digiOutDoesBlend

Indicates that the video digitizer component can blend the resulting image with a matte when displaying the output video. The matte is provided by the application by defining either an `alpha` channel or a mask plane. If this flag is

DigitizerInfo

set to 1, the digitizer component can blend. If this flag is set to 0, the digitizer component does not support this capability.

`digi0OutDoesWarp`

Indicates that the video digitizer component can warp an image when displaying the output video. Warping an image distorts it along one or more axes, perhaps nonlinearly, in effect “bending” the result region. The parameters for the warp operation are specified in the matrix structure for the request. If this flag is set to 1, the digitizer component can warp an image. If this flag is set to 0, the digitizer component does not support this capability.

`digi0OutDoesDMA`

Indicates that the video digitizer component can write to any screen or to offscreen memory. If this flag is set to 1, the digitizer component can use DMA to write to any screen or memory location.

`digi0OutDoesHWPlayThru`

Indicates that the video digitizer component does not need idle time in order to display its video. If this flag is set to 1, your application does not need to grant processor time to the digitizer component at normal display speeds.

`digi0OutDoesILUT`

Indicates that the video digitizer component supports inverse lookup tables for indexed color modes. If this flag is set to 1, the digitizer component uses inverse lookup tables when appropriate.

`digi0OutDoesKeyColor`

Indicates that the video digitizer component supports clipping by means of key colors. If this flag is set to 1, the digitizer component can clip to a region defined by a key color.

`digi0OutDoesAsyncGrabs`

Indicates that the video digitizer component can operate asynchronously. If this flag is set to 1, your application can use `VDSetupBuffers` (III–2055) and `VDGrabOneFrameAsync` (III–2025).

`digi0OutDoesUnreadableScreenBits`

Indicates that the video digitizer may place pixels on the screen that cannot be used when compressing images.

`digi0OutDoesCompress`

Indicates that the video digitizer component supports compressed source devices. These devices provide compressed data directly, without having to use the Image Compression Manager.

`digiOutDoesCompressOnly`

Indicates that the video digitizer component only provides compressed image data; the component cannot provide displayable data. This flag only applies to digitizers that support compressed source devices.

`digiOutDoesPlayThruDuringCompress`

Indicates that the video digitizer component can draw images on the screen at the same time that it is delivering compressed image data. This flag only applies to digitizers that support compressed source devices.

inputCurrentFlags Constants

`digiInSignalLock`

Indicates that the video digitizer component is locked onto the input signal. If this flag is set to 1, the digitizer component detects either vertical or horizontal signal lock.

Discussion

Your application can retrieve information about the capabilities and current status of a video digitizer component. You call `VDGetDigitizerInfo` (III–1998) to retrieve all this information from a video digitizer component. In response, the component formats a `DigitizerInfo` structure. The contents of this structure fully define the capabilities and current status of the video digitizer component.

Associated Functions

`VDGetDigitizerInfo` (III–1998)

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

If you are interested only in the current video digitizer status information, you can call `VDGetCurrentFlags` (III–1996). This function returns the input and output current flags of the video digitizer component.

EditListType

Locates and defines an edit segment within a media.

```
struct EditListType {
    TimeValue    trackDuration;
    TimeValue    mediaTime;
    Fixed        mediaRate;
};
```

EffectsFrameParams

`trackDuration`

The duration of this edit segment in movie time scale units.

`mediaTime`

The starting time within the media of this edit segment, expressed in media time scale units. If -1, it is an empty edit.

`mediaRate`

The relative rate at which to play the media for this edit segment.

Discussion

This is a struct inside the edit list atom, which is part of a movie.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For the atom structure that contains `EditListType` data structures, see the 'elst' (IV-2533) atom.

EffectsFrameParams

Contains information about the current frame of a video effect.

```
struct EffectsFrameParams {
    ICMFrameTimeRecord    frameTime;
    long                  effectDuration;
    Boolean                doAsync;
    unsigned char          pad[3];
    EffectSourcePtr        source;
    void *                 refCon;
};
```

`frameTime`

Timing data for the current frame. This structure includes information such as the total number of frames being rendered in this sequence, and the current frame number.

`effectDuration`

The duration of a single effect frame.

`doAsync`

This field contains `TRUE` if the effect can process asynchronously.

`pad`

Unused.

source

A pointer to the input sources; see the EffectSource (IV–2243) structure.

refCon

A pointer to storage for this instantiation of the effect.

Associated Functions

ImageCodecEffectBegin (II–698)

Programming Info

C interface file: ImageCodec.h

EffectSource

Provides data for the EffectsFrameParams (IV–2242) structure.

```
struct EffectSource {
    long          effectType;
    Ptr           data;
    SourceData    source;
    EffectSourcePtr next;
```

effectType

The type of the effect or a default effect type constant (see below). Enter kEffectRawSource if the source is raw image compression manager data.

data

A pointer to the track data for the effect.

source

The source itself.

next

A pointer to the next source in the input chain.

lastTranslatedFrameTime

The start frame time of last converted frame; this value may be –1.

lastFrameDuration

The duration of the last converted frame; this value may be 0.

lastFrameTimeScale

The time scale of this source frame; this field has meaning only if the lastTranslatedFrameTime and lastFrameDuration fields are valid.

effectType Constants

kEffectRawSource

The source is raw Image Compression Manager data.

EnumListRecord

kEffectGenericType

The source is a generic effect for combining other effects.

'geff'

Value of kEffectGenericType.

Associated Functions

ImageCodecEffectConvertEffectSourceToFormat (II-699)

Programming Info

C interface file: ImageCodec.h

See Also

See the EffectsFrameParams (IV-2242) structure.

EnumListRecord

Stores a constant enumeration as a struct.

```
struct EnumListRecord {  
    long          enumCount;  
    EnumValuePair  values[1];  
};
```

enumCount

Number of enumeration items to follow.

values

Values and names for the enumeration items, packed. See EnumValuePair (IV-2245).

Programming Info

C interface file: ImageCodec.h

EnumRangeRecord

Undocumented

```
struct EnumRangeRecord {  
    long    enumID;  
};
```

enumID

Undocumented

Discussion
Undocumented

Programming Info
C interface file: ImageCodec.h

EnumValuePair

Provides data for an EnumListRecord (IV–2244) structure.

```
struct EnumValuePair {
    long    value;
    Str255  name;
};
```

value
The enumerated value.

name
The corresponding name.

Programming Info
C interface file: ImageCodec.h

See Also
See the EnumListRecord (IV–2244) structure.

EQSpectrumBandsRecord

Undocumented

```
struct EQSpectrumBandsRecord {
    short    count;
    UnsignedFixedPtr frequency;
};
```

count
Undocumented

frequency
Undocumented

Discussion
Undocumented

EventRecord

Programming Info

C interface file: Sound.h

EventRecord

Contains information about a retrieved Mac OS event.

```
struct EventRecord {
    EventKind      what;
    UInt32         message;
    UInt32         when;
    Point          where;
    EventModifiers  modifiers;
};
```

what

A constant (see below) that specifies the kind of event.

message

Additional information (see below) associated with the event. The interpretation of this information depends on the event type.

when

The time when the event was posted, in **ticks** since system startup.

where

For low-level events and operating-system events, this field contains the location of the cursor at the time the event was posted (in global coordinates). For high-level events, it contains a second event specifier, the event ID. The event ID defines the particular type of event within the class of events defined by the message field of the high-level event. For high-level events, you should interpret the where field as having the data type `OSType`, not `Point`.

modifiers

Contains information about the state of the **modifier keys** and the mouse button at the time the event was posted. For activate events, this field also indicates whether the window should be activated or deactivated. In System 7 it also indicates whether the mouse-down event caused your application to switch to the foreground. Each modifier key is represented by a specific bit in the `modifiers` field of the event record structure. The modifier keys include the Option, Command, Caps Lock, Control, and Shift keys. If your application attaches special meaning to any of these keys in combination with other keys or when the mouse button is down, you can test the state of the `modifiers` field to determine the action your application should take. For example, you can use

this information to determine whether the user pressed the Command key and another key to make a menu choice.

what Constants

`nullEvent`

The event code indicating that there are no other pending events. The message field is undefined.

`mouseDown`

The event code indicating that the mouse button has been pressed. The message field is undefined.

`mouseUp`

The event code indicating that the mouse button has been released. The message field is undefined.

`keyDown`

The event code indicating that a key has been pressed. The low-order word of the message field contains the character code and virtual key code, which you can access with the constants `charCodeMask` and `keyCodeMask`, respectively (see below). For Apple Desktop Bus (ADB) keyboards, the low byte of the high-order word contains the ADB address of the keyboard where the keyboard event occurred. The high byte of the high-order word is reserved.

`keyUp`

The event code indicating that a key has been released. The low-order word of the message field contains the character code and virtual key code, which you can access with the constants `charCodeMask` and `keyCodeMask`, respectively (see below). For Apple Desktop Bus (ADB) keyboards, the low byte of the high-order word contains the ADB address of the keyboard where the keyboard event occurred. The high byte of the high-order word is reserved.

`autoKey`

The event code indicating that a key has been repeatedly held down. The low-order word of the message field contains the character code and virtual key code, which you can access with the constants `charCodeMask` and `keyCodeMask`, respectively (see below). For Apple Desktop Bus (ADB) keyboards, the low byte of the high-order word contains the ADB address of the keyboard where the keyboard event occurred. The high byte of the high-order word is reserved.

`updateEvt`

The event code indicating that a window needs updating. The message field contains a pointer to the window to update.

EventRecord

`diskEvt`

The event code indicating that a disk has been inserted. The message field contains the drive number in its low-order word and the File Manager result code in its high-order word.

`activateEvt`

The event code indicating that a window has been activated or deactivated. The message field contains a pointer to the window to activate or deactivate.

`osEvt`

The event code indicating a suspend, resume, or mouse-moved operating-system event.

`kHighLevelEvent`

A high-level event. The message field and the where field of a high-level event define the specific type of high-level event received.

`mouseMovedMessage`

The message code indicating the mouse-moved operating-system event. The `mouseMovedMessage` enumerator is in bits 24–31 of the message field. Bits 2–23 are reserved, and bit 0 and bit 1 are undefined.

`suspendResumeMessage`

The message code indicating a suspend or resume operating-system event. The `suspendResumeMessage` enumerator in bits 24–31 of the message field and a 0 in bit 0 indicate the event is a suspend event. Bit 1 is undefined, and bits 2–23 are reserved.

`resumeFlag`

Flag for a resume event. The `suspendResumeMessage` enumerator in bits 24–31 of the message field and a 1 (the `resumeFlag` enumerator) in bit 0 indicate the event is a resume event. Bit 1 contains a 1 (the `convertClipboardFlag` enumerator) if Clipboard conversion is required, and bits 2–23 are reserved.

`convertClipboardFlag`

Flag indicating that Clipboard conversion is required for a suspend or resume event.

message Mask Constants

`charCodeMask`

Retrieves the character code from the message field; value is 0x000000FF.

`keyCodeMask`

Retrieves the keyboard code from the message field; value is 0x0000FF00.

adbAddrMask

Retrieves the Apple Desktop Bus code from the message field; value is 0x00FF0000.

osEvtMessageMask

Retrieves the Mac OS event code from the message field; value is 0xFF000000.

Associated Functions

ImageCodecIsStandardParameterDialogEvent (II-726)

Programming Info

C interface file: Events.h

See Also

For further information about Mac OS events, see *Inside Macintosh: Macintosh Toolbox Essentials* (listed in the **bibliography**).

ExtComponentResource

Defines the extended structure of a component **resource**, with platform information.

```
struct ExtComponentResource {
    ComponentDescription    cd;
    ResourceSpec            component;
    ResourceSpec            componentName;
    ResourceSpec            componentInfo;
    ResourceSpec            componentIcon;
    long                   componentVersion;
    long                   componentRegisterFlags;
    short                  componentIconFamily;
    long                   count;
    ComponentPlatformInfo   platformArray[1];
};
```

cd

A ComponentDescription (IV-2212) structure that specifies the characteristics of the component.

component

A ResourceSpec (IV-2402) structure that specifies the type and ID of the component code **resource**. The resType field of this structure may contain any value. The component's main entry point must be at offset 0 in the resource.

ExtComponentResource

componentName

A ResourceSpec (IV–2402) structure that specifies the **resource** type and ID for the name of the component. This is a Pascal string. Typically, the component name is stored in a resource of type 'STR'.

componentInfo

A ResourceSpec (IV–2402) structure that specifies the **resource** type and ID for the information string that describes the component. This is a Pascal string. Typically, the information string is stored in a resource of type 'STR'. You might use the information stored in this resource in a Get Info dialog box.

componentIcon

A ResourceSpec (IV–2402) structure that specifies the **resource** type and ID for the icon for a component. Component icons are stored as 32-by-32 bit maps. Typically, the icon is stored in a resource of type 'ICON'. Note that this icon is not used by the Finder; you supply an icon only so that other components or applications can display your component's icon in a dialog box if needed.

componentVersion

The version number of the component. If you specify the `componentDoAutoVersion` flag in `componentRegisterFlags`, the Component Manager must obtain the version number of your component when your component is registered. Either you can provide a version number in your component's **resource**, or you can specify a value of 0 for its version number. If you specify 0, the Component Manager sends your component a version request to get the version number of your component.

componentRegisterFlags

A set of flags (see below) containing additional registration information.

componentIconFamily

The **resource** ID of an icon family. You can provide an icon family in addition to the icon provided in the `componentIcon` field of `ComponentResource` (IV–2219). Note that members of this icon family are not used by the Finder; you supply an icon family only so that other components or applications can display your component's icon in a dialog box if needed.

count

Number of elements in `platformArray`.

platformArray

An array of `ComponentPlatformInfo` (IV–2218) structures.

Programming Info

C interface file: `Components.h`

See Also

See `ComponentResource` (IV–2219), `ComponentResourceExtension` (IV–2221).

ExtendedScheduledSoundHeader

Passes extended sound header fields to a sound channel.

```
struct ExtendedScheduledSoundHeader {
    SoundHeaderUnion    u;
    long                flags;
    short               reserved;
    short               callBackParam1;
    long                callBackParam2;
    TimeRecord          startTime;
    long                recordSize;
    long                extendedFlags;
    long                bufferSize;
};
```

`u`

A `SoundHeaderUnion` used to pass the audio to the channel.

`flags`

Modifier flags (see below).

`reserved`

Reserved. Set to 0.

`callBackParam1`

Passed as parameters to the sound callback associated with the channel.

`callBackParam2`

Passed as parameters to the sound callback associated with the channel.

`startTime`

A `TimeRecord` (IV–2486) structure indicating the time in the future when the sound buffer should play, using the sound clock associated with the channel.

`recordSize`

Holds the size of the field in bytes. In the future this may grow.

`extendedFlags`

Constants (see below) that determine the interpretation of fields in different data structures.

ExtendedSoundComponentData

bufferSize

The buffer size in bytes. This field is not valid if the extendedFlags field's kExtendedSoundBufferSizeValid flag is 0.

flags Constants

kScheduledSoundDoScheduled

Indicates that the startTime field holds the time when this buffer should be played.

kScheduledSoundDoCallback

Indicates that the callback should be called when the buffer completes. This typically is used to chain buffer playbacks together.

extendedFlags Constants

kExtendedSoundSampleCountNotValid

The sample count is not valid. This allows a buffer of audio of bufferSize bytes to not declare the number of audio samples that it will generate. Note that if this flag is set, the bufferSize field must be valid and kExtendedSoundBufferSizeValid must be 1.

kExtendedSoundBufferSizeValid

Indicates if the bufferSize field is valid. If not 1, then you shouldn't use the bufferSize field's value. The QuickTime 4.1 Sound Media Handler passes the buffer size, so this flag is set.

Discussion

The extended forms of the SoundComponentData (IV–2448) and SoundParamBlock (IV–2454) records are normally all with which a sound decompressor implementer need be concerned.

Version Notes

New in QuickTime 4.1.

Programming Info

C interface file: Sound.h

See Also

See ScheduledSoundHeader (IV–2419).

ExtendedSoundComponentData

Provides an extended form of the SoundComponentData (IV–2448) structure.

```
struct ExtendedSoundComponentData {
    SoundComponentData    desc;
    long                  recordSize;
```

```

        long                extendedFlags;
        long                bufferSize;
};

```

desc

A SoundComponentData (IV–2448) structure describing a sound buffer. This may be an ExtendedSoundComponentData (IV–2252) structure. To determine which, check for kExtendedSoundData in the flags field.

recordSize

The number of bytes in the buffer. This allows for cases where the number of bytes isn't derivable directly from the number of samples.

extendedFlags

Constants (see below) that determine the interpretation of fields in different data structures.

bufferSize

Size of the buffer in bytes.

extendedFlags Constants

kExtendedSoundSampleCountNotValid

The sample count is not valid. This allows a buffer of audio of bufferSize bytes to not declare the number of audio samples that it will generate. Note that if this flag is set, the bufferSize field must be valid and kExtendedSoundBufferSizeValid must be 1.

kExtendedSoundBufferSizeValid

Indicates if the bufferSize field is valid. If not 1, then you shouldn't use the bufferSize field's value. The QuickTime 4.1 Sound Media Handler passes the buffer size, so this flag is set.

Discussion

While the ability to not specify the sample count is of little use where the sample references already describes the durations of all the samples, it may prove more useful in other cases. One such case involves sound conversion where the decompressor is asked to generate audio samples until a source buffer is exhausted. Instead of requiring that we know the number of source audio samples in advance, this allows the client to just feed audio until there is no more. Another case involves playback where durations aren't known.

Version Notes

New in QuickTime 4.1.

Programming Info

C interface file: Sound.h

ExtendedSoundParamBlock

See Also

See SoundComponentData (IV–2448).

ExtendedSoundParamBlock

Provides an extended form of SoundParamBlock (IV–2454).

```
struct ExtendedSoundParamBlock {
    SoundParamBlock    pb;
    short              reserved;
    long               extendedFlags;
    long               bufferSize;
};
```

pb

A SoundParamBlock (IV–2454) structure. Measuring its recordSize field lets you tell if it is an ExtendedSoundParamBlock (IV–2254) structure. Since the embedded SoundParamBlock structure contains a SoundComponentData field, the kExtendedSoundSampleCountNotValid flag set refers to the sampleCount field of that structure. The flags field of the embedded SoundComponentData structure should never have its kExtendedSoundData flag set. This would mislead a client passed only a pointer to SoundComponentData structure to misinterpret it as an extended structure.

reserved

Reserved; ignore.

extendedFlags

Constants (see below) that determine the interpretation of fields in different data structures.

bufferSize

Size of the buffer in bytes.

extendedFlags Constants

kExtendedSoundSampleCountNotValid

The sample count is not valid. This allows a buffer of audio of bufferSize bytes to not declare the number of audio samples that it will generate. Note that if this flag is set, the bufferSize field must be valid and kExtendedSoundBufferSizeValid must be 1.

kExtendedSoundBufferSizeValid

Indicates if the bufferSize field is valid. If not 1, then you shouldn't use the bufferSize field's value. The QuickTime 4.1 Sound Media Handler passes the buffer size, so this flag is set.

Discussion

An `ExtendedSoundParamBlock` structure can be passed to a sound component's `SoundComponentPlaySourceBuffer` (III–1854) routine. Also, the `moreRtn` of the `SoundParamBlock` (IV–2454) structure may return a reference to an `ExtendedSoundParamBlock` structure.

Version Notes

New in QuickTime 4.1.

Programming Info

C interface file: `Sound.h`

See Also

See `SoundParamBlock` (IV–2454).

ExtSoundHeader

Stores sampled-sound data that is more complex than a `SoundHeader` (IV–2452) structure can describe.

```
struct ExtSoundHeader {
    Ptr          samplePtr;
    unsigned long numChannels;
    UnsignedFixed sampleRate;
    unsigned long loopStart;
    unsigned long loopEnd;
    UInt8        encode;
    UInt8        baseFrequency;
    unsigned long numFrames;
    extended80   AIFFSampleRate;
    Ptr          markerChunk;
    Ptr          instrumentChunks;
    Ptr          AESRecording;
    unsigned short sampleSize;
    unsigned short futureUse1;
    unsigned long  futureUse2;
    unsigned long  futureUse3;
    unsigned long  futureUse4;
    UInt8          sampleArea[1];
};
```

`samplePtr`

A pointer to the sampled-sound data. If the sampled sound is located in memory immediately after the `futureUse4` field, then this field should be set to `NIL`. Otherwise, this field is a pointer to the memory location of the sampled-sound data.

ExtSoundHeader

numChannels

The number of channels in the sample.

sampleRate

The sample rate at which the frames were sampled before compression; see “Sound Sample Rates” (IV–2695).

loopStart

The starting point of the portion of the extended sampled sound header that is to be used by the Sound Manager. These loop points specify the byte numbers in the sampled data to be used as the beginning and end points to cycle through when playing the sound. The loop starting and ending points are 0-based.

loopEnd

The end of the loop points of the sound before compression.

encode

The method of encoding (if any) used to generate the sampled-sound data (see below). For a compressed sound header, you should specify the constant `cmpSH`. Encode option values in the ranges 0 through 63 and 128 to 255 are reserved for use by Apple. You are free to use numbers in the range 64 through 127 for your own encode options.

baseFrequency

The pitch of the original sampled sound.

numFrames

The number of frames in the sampled-sound data. Each frame contains `numChannels` bytes for 8-bit sound data.

AIFFSampleRate

The sample rate at which the frames were sampled before compression, as expressed in the 80-bit extended data type representation (IEEE sample rate).

markerChunk

Synchronization information. The `markerChunk` field is not presently used and should be set to `NIL`.

instrumentChunks

AIFF Instrument information.

AESRecording

Information related to audio recording devices.

sampleSize

The number of bits in each sample frame.

futureUse1

Reserved.

futureUse2

Reserved.

futureUse3

Reserved.

futureUse4

The four `futureUse` fields are reserved for use by Apple. To maintain compatibility with future versions of system software, you should always set these fields to 0.

sampleArea

An array of interleaved sample points, each of which contains a value similar to the values in a wave-table description. For 8-bit sampled-sound data, these values are interpreted as offset values, where 0x80 represents an amplitude of 0. The value 0x00 is the largest negative amplitude, and 0xFF is the largest positive amplitude.

encode Constants

cmpSH

Compressed sound header; value is 0xFE.

extSH

Extended sound header; value is 0xFF.

stdSH

Standard sound header; value is 0x00.

Discussion

Most of the fields of the extended sound header correspond to fields of the sampled sound header. However, the extended sound header allows the encoding of stereo sound. The `numChannels` field contains the number of channels of sound recorded, and the `numFrames` field contains the number of frames of sound recorded in each channel. To compute the total number of bytes of a sample, multiply the values in the `numChannels`, `numFrames`, and `sampleSize` fields and divide by the number of bytes per sample (typically 8 or 16).

Version Notes

Although `ExtSoundHeader` and `CmpSoundHeader` (IV–2176) support the storage of 16-bit sound, only versions 3.0 and later of the Sound Manager can play 16-bit sounds. If your application uses 16-bit sound, you must convert it to 8-bit sound before earlier versions of the Sound Manager can play it.

Programming Info

C interface file: `Sound.h`

See Also

See `CmpSoundHeader` (IV–2176).

FieldInfoImageDescriptionExtension

FieldInfoImageDescriptionExtension

Undocumented

```
struct FieldInfoImageDescriptionExtension {  
    UInt8    fieldCount;  
    UInt8    fieldOrderings;  
};
```

fieldCount

Undocumented

fieldOrderings

Undocumented

Discussion

Undocumented

Programming Info

C interface file: ImageCodec.h

FieldInfoImageDescriptionExtension2

Remaps to FieldInfoImageDescriptionExtension (IV–2258).

```
struct FieldInfoImageDescriptionExtension2 {  
    UInt8    fields;  
    UInt8    detail;  
};
```

fields

Undocumented

detail

Undocumented

Discussion

Undocumented

Programming Info

C interface file: ImageCodec.h

FixedPoint

Defines the position of a geometric point in fixed-point numbers.


```
struct FixedPoint {
    Fixed    x;
    Fixed    y;
};
```

x

The x (horizontal) coordinate of the point.

y

The y (vertical) coordinate of the point.

Special Considerations

This function differs from the `Point` (IV–2339) function in that its parameters are fixed-point numbers and the order of horizontal and vertical parameters is reversed.

Associated Functions

`CurveGetNearestPathPoint` (I–156)

Programming Info

C interface file: `MacTypes.h`

FixedRangeRecord

Undocumented.

```
struct FixedRangeRecord {
    Fixed    minValue;
    Fixed    maxValue;
    Fixed    scaleValue;
    long     precisionDigits;
};
```

minValue

Undocumented

maxValue

Undocumented

scaleValue

Undocumented

precisionDigits

Undocumented

Discussion

Undocumented

FixedRect

Programming Info

C interface file: ImageCodec.h

FixedRect

Defines the size and location of a rectangle in fixed-point numbers.

```
struct FixedRect {  
    Fixed    left;  
    Fixed    top;  
    Fixed    right;  
    Fixed    bottom;  
};
```

left

The x coordinate of the upper-left corner of the rectangle.

top

The y coordinate of the upper-left corner of the rectangle.

right

The x coordinate of the lower-right corner of the rectangle.

bottom

The y coordinate of the lower-right corner of the rectangle.

Special Considerations

This structure differs from the Rect (IV–2399) structure in that its parameters are fixed-point numbers and their order is different.

Associated Functions

TransformFixedRect (III–1952)

Programming Info

C interface file: MacTypes.h

See Also

See *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

FlashDescription

Undocumented

```
struct FlashDescription {  
    long    descSize;  
    long    dataFormat;
```

```

    long    resvd1;
    short   resvd2;
    short   dataRefIndex;
    long    version;
    OSType  decompressorType;
    long    flags;
};

```

descSize

Undocumented

dataFormat

Undocumented

resvd1

Undocumented

resvd2

Undocumented

dataRefIndex

Undocumented

version

Undocumented

decompressorType

The decompressor to use; 0 for no decompression.

flags

Undocumented

Discussion

Undocumented

Programming Info

C interface file: `Movies.h`

See Also

Undocumented

FontInfo

Contains dimensional information for the current font.

```

struct FontInfo {
    short   ascent;
    short   descent;
    short   widMax;
};

```

FSSpec

```
    short    leading;  
};
```

ascent

The measurement in pixels from the baseline to the ascent line of the font.

descent

The measurement in pixels from the baseline to the descent line of the font.

widMax

The width in pixels of the widest glyph in the font.

leading

The measurement in pixels from the descent line to the ascent line below it.

Associated Functions

QDTxMeasProc (III–2123)

Programming Info

C interface file: QuickdrawText.h

FSSpec

Identifies a Mac OS file or directory.

```
struct FSSpec {  
    short    vRefNum;  
    long     parID;  
    StrFileName    name;  
};
```

vRefNum

Volume reference number.

parID

Directory ID of parent directory.

name

Filename or directory name; a Str63 string on the Mac OS.

Discussion

The FSSpec structure provides a simple and standard format for specifying files and directories. You can pass that specification directly to any file-manipulation routines that accept FSSpec records.

Associated Functions

ConvertMovieToFile (I–134)

Programming Info

C interface file: Files.h

GCInstrumentData

Provides data for the GCPart (IV–2263) structure.

```
struct GCInstrumentData {
    ToneDescription  tone;
    long            knobCount;
    long            knob[1];
};
```

tone

A ToneDescription (IV–2487) structure.

knobCount

The number of InstKnobRec structures in the list.

knob

An array of pointers to InstKnobRec (IV–2293) structures.

Programming Info

C interface file: QuickTimeMusic.h

GCPart

Defines a part in the QuickTime Music Architecture.

```
struct GCPart {
    long            hwInstrumentNumber;
    short          controller[128];
    long           volume;
    long           polyphony;
    long           midiChannel;
    GCInstrumentData id;
};
```

hwInstrumentNumber

The instrument number of the instrument for the part.

controller

An array of 128 bits identifying the available controllers; see “Music Controllers” (IV–2682). Bits are numbered from 1 to 128, starting with the most

GDevice

significant bit of the long word and continuing to the least significant of the last bit.

volume

The sound volume for this part, ranging from -1.0 to $+1.0$. The high-order 8 bits contain the integer part; the low-order 8 bits contain the fractional part. A value of $+1.0$ constitutes the maximum volume of the user's computer. Negative values are silent but retain the magnitude of the volume setting.

polyphony

The maximum number of voices.

midiChannel

The system MIDI channel or, for a hardware device, the slot number.

id

A `GCInstrumentData` (IV-2263) structure.

Associated Functions

`MusicDerivedSetInstrument` (II-976)

Programming Info

C interface file: `QuickTimeMusic.h`

GDevice

Stores state information for video devices and offscreen graphics worlds.

```
struct GDevice {
    short      gdRefNum;
    short      gdID;
    short      gdType;
    ITabHandle gdITable;
    short      gdResPref;
    SProcHndl  gdSearchProc;
    CProcHndl  gdCompProc;
    short      gdFlags;
    PixMapHandle gdPMap;
    long       gdRefCon;
    GDHandle    gdNextGD;
    Rect        gdRect;
    long       gdMode;
    short      gdCCBytes;
    short      gdCCDepth;
    Handle      gdCCXData;
    Handle      gdCCXMask;
```

gdRefNum

The reference number of the driver for the screen associated with the video device. For most video devices, this information is set at system startup time.

gdID

Reserved. If you create your own GDevice structure, set this field to 0.

gdType

A constant (see below) that identifies the general type of graphics device.

gdITable

A handle to the inverse table for color mapping.

gdResPref

The preferred resolution for inverse tables.

gdSearchProc

A handle to the list of search functions. For details, see the chapter “Color Manager” in *Advanced Color Imaging on the Mac OS* (listed in the **bibliography**). Its value is NIL for the default function.

gdCompProc

A handle to a list of complement functions; see CProcRec (IV-2225).

gdFlags

Flag bits (see below) that describe the GDevice structure’s attributes. To set the attribute bits in this field, use SetDeviceAttribute, declared in the file Quickdraw.h; do not set these flags directly in the GDevice structure.

gdPMap

A handle to a PixMap (IV-2332) structure giving the dimension of the image buffer, along with the characteristics of the graphics device (resolution, storage format, color depth, and color table).

gdRefCon

A value used by system software to pass device-related parameters. Since a graphics device is shared, you shouldn’t store data here.

gdNextGD

A handle to the next graphics device in the device list. If this is the last graphics device in the device list, the field contains 0.

gdRect

The boundary rectangle of the graphics device represented by the GDevice structure. The main screen has the upper-left corner of the rectangle set to (0,0). All other graphics devices are relative to this point.

GDevice

gdMode

The current setting for the graphics device mode. This value is passed to the video driver to set its pixel depth and to specify color or black and white; applications don't need this information.

gdCCBytes

The rowBytes value of the expanded cursor. Your application should not change this field.

gdCCDepth

The depth of the expanded cursor. Your application should not change this field.

gdCCXData

A handle to the cursor's expanded data. Your application should not change this field.

gdCCXMask

A handle to the cursor's expanded mask. Your application should not change this field.

gdReserved

Reserved for future expansion; it must be set to 0 for future compatibility.

gdExt

QuickTime 3.0 private data.

gdType Constants

clutType

CLUT device; that is, one whose colors are mapped with a **color lookup table**.

fixedType

Fixed colors; that is, the **color lookup table** can't be changed.

directType

Direct RGB colors.

gdFlags Constants

gdDevType

If this bit is set to 0, the graphics device is black and white; if it is set to 1, the graphics device supports color.

burstDevice

If this bit is set to 1, the graphics device supports block transfers.

ext32Device

If this bit is set to 1, the graphics device must be used in 32-bit mode.

ramInit

If this bit is set to 1, the graphics device has been initialized from RAM.

`mainScreen`

If this bit is set to 1, the graphics device is the main screen.

`allInit`

If this bit is set to 1, all graphics devices were initialized from 'scrn' **resources**.

`screenDevice`

If this bit is set to 1, the graphics device is a screen.

`noDriver`

If this bit is set to 1, the GDevice structure has no driver.

`screenActive`

If this bit is set to 1, the graphics device is active.

Discussion

When Mac OS starts up, it allocates and initializes one handle to a GDevice structure for each video device it finds. When you create a new offscreen graphics world, Color QuickDraw automatically creates a GDevice structure for it. The system links these GDevice records in a list, called the device list. By default, the GDevice structure corresponding to the first video device found is marked as the current device; all other graphics devices in the list are initially marked as inactive.

Special Considerations

Your application should never need to change the fields of a GDevice structure directly.

Programming Info

C interface file: Quickdraw.h

GenericKnobDescription

Describes a knob for the generic music component.

```
struct GenericKnobDescription {
    KnobDescription    kd;
    long               hw1;
    long               hw2;
    long               hw3;
    long               settingsID;
};
```

`kd`

A KnobDescription (IV-2299) structure.

`hw1`

Undocumented

GenericKnobDescriptionList

hw2

Undocumented

hw3

Undocumented

settingsID

Undocumented

Discussion

Undocumented

Associated Functions

MusicDerivedSetKnob (II–976)

Programming Info

C interface file: QuickTimeMusic.h

GenericKnobDescriptionList

Provides a list of GenericKnobDescription (IV–2267) structures.

```
struct GenericKnobDescriptionList {  
    long                knobCount;  
    GenericKnobDescription knob[1];  
};
```

knobCount

The number of GenericKnobDescription structures.

knob

An array of GenericKnobDescription structures.

Associated Functions

MusicGenericGetKnobList (II–984)

Programming Info

C interface file: QuickTimeMusic.h

GetMovieCompleteParams

Defines the layout of the complete movie parameter structure used by MediaInitialize (II–877).

```
struct GetMovieCompleteParams {  
    short                version;
```

```

Movie          theMovie;
Track          theTrack;
Media          theMedia;
TimeScale      movieScale;
TimeScale      mediaScale;
TimeValue      movieDuration;
TimeValue      trackDuration;
TimeValue      mediaDuration;
Fixed          effectiveRate;
TimeBase       timeBase;
short          volume;
Fixed          width;
Fixed          height;
MatrixRecord   trackMovieMatrix;
CGrafPtr       moviePort;
GDHandle       movieGD;
PixMapHandle   trackMatte;
QTAtomContainer inputMap;
};

```

version

Specifies the version of this structure. This field is always set to 0.

theMovie

Identifies the movie that contains the current media's track. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain information about the movie that is using your media.

theTrack

Identifies the track that contains the current media. This track identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain information about the track that contains your media. For example, you might call `GetTrackNextInterestingTime` (I-537) to examine the track's edit list.

theMedia

Identifies the current media. This media identifier is supplied by the Movie Toolbox. Your derived media handler can use this identifier to read samples or sample descriptions from the current media, using `GetMediaSample` (I-443) and `GetMediaSampleDescription` (I-445).

movieScale

Specifies the time scale of the movie that contains the current media's track. If the Movie Toolbox changes the movie's time scale, the toolbox calls your derived media handler's `MediaSetMovieTimeScale` (II-899) function.

GetMovieCompleteParams

mediaScale

Specifies the time scale of the current media. If the Movie Toolbox changes your media's time scale, the toolbox calls your derived media handler's `MediaSetMediaTimeScale` (II-898) function.

movieDuration

Contains the movie's duration. This value is expressed in the movie's time scale.

trackDuration

Contains the track's duration. This value is expressed in the movie's time scale.

mediaDuration

Contains the media's duration. This value is expressed in the media's time scale.

effectiveRate

Contains the media's effective rate. This rate ties the media's time scale to the passage of absolute time, and does not necessarily correspond to the movie's rate. This value takes into account any master time bases that may be serving the media's time base. The value of this field indicates the number of time units (in the media's time scale) that pass each second. This rate is represented as a 32-bit, fixed-point number. The high-order 16 bits contain the integer portion, and the low-order 16 bits contain the fractional portion. The rate is negative when time is moving backward for the media. Whenever the Movie Toolbox changes your media's effective rate, it calls your derived media handler's `MediaSetRate` (II-903) function.

timeBase

Identifies the media's time base.

volume

Contains the media's current volume setting. This value is represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer portion; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. If QuickTime changes your media's volume, it calls your derived media handler's `MediaGSetVolume` (II-871) function.

width

Indicates the width, in pixels, of the track rectangle. This field, along with the height field, specifies a rectangle that surrounds the image that is displayed when the current media is played. This value corresponds to the x coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number. If the Movie Toolbox modifies this rectangle, the toolbox calls your derived media handler's `MediaSetDimensions` (II-891) function. Note that your

media need not present only a rectangular image. The Movie Toolbox can use a clipping region to cause your media's image to be displayed in a region of arbitrary shape, and it can use a matte to control the image's transparency. The toolbox calls your derived media handler's `MediaSetClip` (II-890) function whenever it changes your media's clipping region. The `trackMatte` field in this structure specifies a matte region.

`height`

Indicates the height, in pixels, of the track rectangle. This value corresponds to the y coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number.

`trackMovieMatrix`

Specifies the matrix that transforms your media's pixels into the movie's coordinate system. The Movie Toolbox obtains this matrix by concatenating the track matrix and the movie matrix. You should use this matrix whenever you are displaying graphical data from your media. Whenever the Movie Toolbox modifies this matrix, it calls your derived media handler's `MediaSetMatrix` (II-897) function.

`moviePort`

Indicates the movie's graphics port. Whenever the Movie Toolbox changes the movie's graphics world, it calls your derived media handler's `MediaSetGWorld` (II-893) function.

`movieGD`

Specifies the movie's graphics device. Whenever the Movie Toolbox changes the movie's graphics world, it calls your derived media handler's `MediaSetGWorld` (II-893) function.

`trackMatte`

Identifies the matte region assigned to the track that uses your media. This field contains a handle to a pixel map that contains a blend matte. Your component is not responsible for disposing of this matte. If there is no matte, this field is set to `NIL`.

`inputMap`

A reference to the media's input map. The media input map should not be modified or disposed.

Associated Functions

`MediaInitialize` (II-877)

Programming Info

C interface file: `MediaHandlers.h`

GMInstrumentInfo

See Also

See `MediaInitialize` (II–877).

GMInstrumentInfo

Provides information about a General MIDI instrument within an instrument component.

```
struct GMInstrumentInfo {  
    long    cmpInstID;  
    long    gmInstNum;  
    long    instMatch;  
};
```

`cmpInstID`

The number of the instrument within the instrument component.

`gmInstNum`

The General MIDI, or standard, instrument number.

`instMatch`

A flag (see below) indicating how the instrument matches the requested instrument.

instMatch Constants

`kInstrumentExactMatch`

The instrument exactly matches the request.

`kInstrumentRecommendedSubstitute`

The instrument is the approved substitute.

`kInstrumentQualityField`

The high-order 8 bits of this field specify the quality of the selected instrument. Higher values specify higher quality.

`kRoland8BitQuality`

For built-in instruments, the value of the high-order 8 bits is always

`kInstrumentRoland8BitQuality`.

`kInstrumentRoland8BitQuality`

The quality of an 8-bit Roland instrument.

Programming Info

C interface file: `QuickTimeMusic.h`

GradientColorRecord

Stores color gradient information.

```
struct GradientColorRecord {
    ARGBColor    thisColor;
    Fixed        endingPercentage;
};
```

thisColor

Specifies a color used for a gradient.

endingPercentage

Specifies the percentage of the gradient (expressed as value between 0 and 1, where 0 is the beginning of the gradient) at which the specified color begins.

Programming Info

C interface file: ImageCodec.h

GrafPort

Defines a complete drawing environment for black-and-white graphics operations.

```
struct GrafPort {
    short          device;
    BitMap         portBits;
    Rect           portRect;
    RgnHandle      visRgn;
    RgnHandle      clipRgn;
    Pattern        bkPat;
    Pattern        fillPat;
    Point          pnLoc;
    Point          pnSize;
    short          pnMode;
    Pattern        pnPat;
    short          pnVis;
    short          txFont;
    StyleField     txFace;
    short          txMode;
    short          txSize;
    Fixed          spExtra;
    long           fgColor;
    long           bkColor;
    short          colrBit;
    short          patStretch;
    Handle         picSave;
```

GrafPort

```
    Handle      rgnSave;  
    Handle      polySave;  
    QDProcsPtr  grafProcs;  
};
```

device

See CGrafPort (IV–2168).

portBits

See CGrafPort (IV–2168). In a GrafPort structure, this field contains a complete 14-byte BitMap (IV–2164) structure. In a CGrafPort structure, this field is partly replaced by the 4-byte portPixMap field, which contains a handle to a PixMap structure. In what would be the rowBytes field of the BitMap structure, a CGrafPort structure has a 2-byte portVersion field in which the two high bits are always set to 1. QuickTime uses these bits to distinguish CGrafPort records from GrafPort records, in which the two high bits of the rowBytes field are always 0. Following the portBits field in the CGrafPort structure are the portVersion and grafVars fields. The grafVars field contains a handle to a GrafVars structure; this handle is not included in the GrafPort structure. For information about the GrafVars structure, see *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

portRect

See CGrafPort (IV–2168).

visRgn

See CGrafPort (IV–2168).

clipRgn

See CGrafPort (IV–2168).

bkPat

In a GrafPort structure, the bkPat, pnPat, and fillPat fields hold 8-byte bit patterns. In a CGrafPort (IV–2168) structure, these fields are partly replaced by three 4-byte handles to pixel patterns. The resulting 12 bytes of additional space in the CGrafPort structure are taken up by the rgbFgColor and rgbBkColor fields, which contain 6-byte RGBColor (IV–2403) structures specifying the optimal foreground and background colors for the color graphics port. Note that the closest matching available colors, which QuickTime actually uses to render the foreground and background, are stored in the fgColor and bkColor fields of the CGrafPort structure.

fillPat

See the bkPat field (above).

pnLoc

See CGrafPort (IV–2168).

pnSize

See CGrafPort (IV–2168).

pnMode

See CGrafPort (IV–2168).

pnPat

See the bkPat field (above).

pnVis

See CGrafPort (IV–2168).

txFont

See CGrafPort (IV–2168).

txFace

The character style of the text, with values from the set defined by the Style type, which includes such styles as bold, italic, and shaded. You can apply stylistic variations either alone or in combination. This field is initially set to plain text.

txMode

See CGrafPort (IV–2168).

txSize

See CGrafPort (IV–2168).

spExtra

See CGrafPort (IV–2168).

fgColor

See CGrafPort (IV–2168).

bkColor

See CGrafPort (IV–2168).

colrBit

See CGrafPort (IV–2168).

patStretch

See CGrafPort (IV–2168).

picSave

See CGrafPort (IV–2168).

rgnSave

See CGrafPort (IV–2168).

gxPath

polySave

See CGrafPort (IV–2168).

grafProcs

See CGrafPort (IV–2168). In a GrafPort structure, you can supply this field with a pointer to a QDProcs (IV–2342) structure; in a CGrafPort structure, you provide this field with a pointer to a CQDProcs (IV–2226) structure.

Discussion

See CGrafPort (IV–2168).

Version Notes

The GrafPort structure has been largely superseded by the CGrafPort (IV–2168) structure, which defines a full color environment. The two structures are the same size; QuickTime distinguishes between them by examining the portBits field.

Programming Info

C interface file: Quickdraw.h

See Also

See CGrafPort (IV–2168).

gxPath

Encapsulates a path contour geometry.

```
struct gxPath {
    long      vectors;
    long      controlBits[1];
    gxPoint   vector[1];
};
```

vectors

The number of geometric points in the contour.

controlBits

Bit flags that indicate which geometric points are on curve and which are off-curve control points.

vector

The coordinates of the geometric points.

Discussion

A path contour is a series of connected points. The contour can contain both straight lines and curves. Therefore, the geometric points that make up a path contour can

be on-curve points or off-curve control points. The path shape allows you to group any number of contours within a single QuickDraw GX shape.

Programming Info

C interface file: `GXTypes.h`

gxPaths

Encapsulates a multiple-path geometry.

```
struct gxPaths {  
    long    contours;  
    gxPath  contour[1];  
};
```

`contours`

The number of path contours.

`contour`

The path contours; see `gxPath` (IV–2276).

Discussion

The `contours` field indicates the total number of contours (in other words, the total number of separate paths), and the `contour` field is an array that contains the path geometries. Since a `gxPaths` structure is of variable length and every element in it is of type `long`, you can define a path geometry as an array of long integer values.

Associated Functions

`CurveCountPointsInPath` (I–153)

Programming Info

C interface file: `GXTypes.h`

See Also

See `gxPath` (IV–2276).

gxPoint

Defines a point in vector graphics.

```
struct gxPoint {  
    Fixed    x;  
    Fixed    y;  
};
```

ICMAlignmentProcRecord

x

A horizontal distance. Greater values of the x field indicate distances further to the right.

y

A vertical distance. Greater values of the y field indicate distances further down.

Discussion

The location of the origin depends on the context where you use the point; for example, it might be the upper-left corner of a view port. Notice that the x and y fields are of type Fixed. QuickDraw GX allows you to specify fractional coordinate positions.

Associated Functions

CurveGetPathPoint (I-157)

Programming Info

C interface file: GXTypes.h

ICMAlignmentProcRecord

Specifies an image compression alignment callback.

```
struct ICMAlignmentProcRecord {  
    ICMAlignmentUPP    alignmentProc;  
    long                alignmentRefCon;  
};
```

alignmentProc

Contains a **Universal Procedure Pointer** that accesses your ICMAlignmentProc (III-2086) callback.

alignmentRefCon

Contains a reference constant for use by your callback.

Discussion

This structure defines a pointer to an alignment function. You assign an alignment function by passing a pointer to this structure.

Associated Functions

AlignScreenRect (I-46)

Programming Info

C interface file: ImageCompression.h

ICMCompletionProcRecord

Specifies an image compression completion callback.

```
struct ICMCompletionProcRecord {
    ICMCompletionUPP    completionProc;
    long                completionRefCon;
};
```

completionProc

Contains a **Universal Procedure Pointer** that accesses your ICMCompletionProc (III–2087) callback.

completionRefCon

Contains a reference constant for use by your callback.

Discussion

This structure governs whether you perform a compression asynchronously. If the completionProc field in this structure is set to NIL, perform the compression synchronously. If this field is not NIL, it specifies an application completion function. Perform the compression asynchronously and call that completion function when your component is finished. If the completionProc field in this structure has a value of –1, perform the operation asynchronously but do not call the application’s completion function

Associated Functions

CompressSequenceFrame (I–124)

Programming Info

C interface file: ImageCompression.h

See Also

See CodecCompressParams (IV–2184). For information on calling completion functions, see the chapter “Image Compression Manager” in *Inside Macintosh: QuickTime* (listed in the **bibliography**).

ICMDataProcRecord

Specifies an image compression data-loading function.

```
struct ICMDataProcRecord {
    ICMDataUPP    dataProc;
    long          dataRefCon;
};
```

ICMFlushProcRecord

`dataProc`

Contains a pointer to your data-loading function.

`dataRefCon`

Contains a reference constant for use by your data-loading function.

Discussion

If there is no data-loading function, the Image Compression Manager sets the `dataProc` field to `NIL`, and the entire image must be in memory at the location specified by the `codecData` field of the `ImageSubCodecDecompressRecord` (IV–2289) structure.

Associated Functions

`FDecompressImage` (I–355)

Programming Info

C interface file: `ImageCompression.h`

ICMFlushProcRecord

Specifies an image compression data-unloading callback.

```
struct ICMFlushProcRecord {
    ICMFlushUPP    flushProc;
    long           flushRefCon;
};
```

`flushProc`

Contains a pointer to your data-unloading function.

`flushRefCon`

Contains a reference constant for use by your data-unloading function.

Discussion

If there is not enough memory to store a compressed image, your application may provide a function that unloads some of the compressed data. This field contains a structure that identifies that data-unloading function. If the application did not provide a data-unloading function, the `flushProc` field in this structure is set to `NIL`. In this case, your component writes the entire compressed image into the memory location specified by the `data` field.

Associated Functions

`FCompressImage` (I–344)

Programming Info

C interface file: `ImageCompression.h`

See Also

See `CodecCompressParams` (IV–2184). See the chapter “Image Compression Manager” in *Inside Macintosh: QuickTime* (listed in the **bibliography**) for more information about data-unloading functions.

ICMFrameTimeInfo

Contains timing information about the display of a frame.

```
struct ICMFrameTimeInfo {
    wide    startTime;
    long    scale;
    long    duration;
};
```

`startTime`

The start time of the frame, expressed in media time scale units from the beginning of the track.

`scale`

The media time scale in units per second.

`duration`

The frame’s duration in media time scale units.

Programming Info

C interface file: `ImageCodec.h`

See Also

See the `CodecDecompressParams` (IV–2190) structure.

ICMFrameTimeRecord

Contains a frame’s time information for scheduled asynchronous decompression operations.

```
struct ICMFrameTimeRecord {
    wide    value;
    long    scale;
    void *   base;
    long    duration;
    Fixed    rate;
    long    recordSize;
    long    frameNumber;
    long    flags;
```

ICMPixelFormatInfo

```
        wide      virtualStartTime;  
        long      virtualDuration;  
};
```

value

Specifies the time at which the frame is to be displayed.

scale

Indicates the units for the frame's display time.

base

Refers to the time base.

duration

Specifies the duration for which the frame is to be displayed. This must be in the same units as specified by the `scale` field. It is 0 if the duration is unknown.

rate

Indicates the time base's effective rate.

recordSize

Total number of bytes in this structure.

frameNumber

Number of frame; 0 if the frame number is not known.

flags

Flag (see below) to indicate if `virtualStartTime` and `virtualDuration` are valid.

virtualStartTime

Conceptual start time.

virtualDuration

Conceptual duration.

flags Constants

`icmFrameTimeHasVirtualStartTimeAndDuration`

Indicates that `virtualStartTime` and `virtualDuration` are valid.

Programming Info

C interface file: `Movies.h`

See Also

See the `CodecDecompressParams` (IV–2190) structure.

ICMPixelFormatInfo

Defines a pixel format.


```

struct ICMPixelFormatInfo {
    long        size;
    unsigned long    formatFlags;
    short       bitsPerPixel[14];
};

```

size

The size of this structure. This quantity isn't necessarily equal to `sizeof(ICMPixelFormatInfo)` because it is dependent on whether the pixel format is chunky or planar, and, if planar, the number of components (see below).

formatFlags

A constant (see below) indicating the pixel format.

bitsPerPixel

An array that defines the number of bits for each component. The element `bitsPerPixel[0]` contains the number of bits for the first component, `bitsPerPixel[1]` the number of bits for the second component, etc. The meaning of this parameter depends on the format flag (see below).

formatFlags Constants**kICMPixelFormatIsPlanarMask**

If this flag is 1, the pixel format is a planar mask and `bitsPerPixel[]` represents the bits for each pixel component. If this flag is 0, the pixel format is chunky (not planar) and `bitsPerPixel[0]` represents the bits per pixel. Chunky pixel formats pack the different components together. For example, 3 pixels of 32-bit ARGB is represented in memory as ARGBARGBARGB. Planar formats pack the different components separately. If the pixel format is planar, then `(formatFlags & kICMPixelFormatIsPlanarMask)` is equal to the number of components.

kICMPixelFormatIsIndexed

If the pixel format is indexed (which, by definition, means that there are no individual components) then this flag is 1. Generally, color modes of 8 bit per pixel or less are indexed.

kICMPixelFormatIsSupportedByQD

If this flag is 1, you can call `QuickDraw` on `PixMap` (IV-2332) structures that store this kind of pixel data. With Macintosh, the classic QD pixel formats will have this set, but not any of the YUV pixel formats. With Windows, more formats will have this set, because the Windows implementation of `QuickDraw` needs to support more pixel formats.

ICMProgressProcRecord

Discussion

You can represent a format that has from 1 to 14 discrete components using this data structure. For ARGB, there are 4 components. RGB without an alpha channel has 3 components. A component count of 15 is reserved for future expansion.

Associated Functions

ICMGetPixelFormatInfo (II-673)

Programming Info

C interface file: ImageCompression.h

ICMProgressProcRecord

Specifies an image compression progress callback.

```
struct ICMProgressProcRecord {  
    ICMProgressUPP    progressProc;  
    long              progressRefCon;  
};
```

progressProc

Contains a pointer to your **progress function**.

progressRefCon

Contains a reference constant for use by your **progress function**.

Discussion

During a compression operation, your compressor may occasionally call a function that the application provides in order to report your progress. This field contains a structure that identifies the **progress function**. If the progressProc field in this structure is set to NIL, the application has not supplied a progress function

Associated Functions

DrawPictureFile (I-310)

Programming Info

C interface file: ImageCompression.h

See Also

See CodecCompressParams (IV-2184). See the chapter “Image Compression Manager” in *Inside Macintosh: QuickTime* (listed in the **bibliography**) for more information about **progress functions**.

ImageDescription

Defines the characteristics of a compressed image or sequence.

```
struct ImageDescription {
    long        idSize;
    CodecType    cType;
    long        resvd1;
    short       resvd2;
    short       dataRefIndex;
    short       version;
    short       revisionLevel;
    long        vendor;
    CodecQ      temporalQuality;
    CodecQ      spatialQuality;
    short       width;
    short       height;
    Fixed       hRes;
    Fixed       vRes;
    long        dataSize;
    short       frameCount;
    Str31       name;
    short       depth;
    short       clutID;
};
```

idSize

Defines the total size of this image description structure with extra data including **color lookup tables** and other per-sequence data.

cType

Indicates the type of compressor component that created this compressed image data. The value of this field indicates the compression algorithm supported by the component. The Codec data type defines a field in the compressor name list structure that identifies the compression method employed by a given compressor component. Apple assigns these values so that they remain unique. These values correspond, in turn, to text strings that can identify the compression method to the user. See “Codec Identifiers” (IV-2655) for a list of values.

resvd1

Reserved; set to 0.

resvd2

Reserved; set to 0.

ImageDescription

dataRefIndex

Reserved; set to 0.

version

Indicates the version of the compressed data. The contents of this field should indicate the version of the compression algorithm that was used to create the compressed data. By examining this field, decompressors that support many versions of an algorithm can determine the proper way to decompress the image.

revisionLevel

Indicates the version of the compressor that created the compressed image. Developers of compressors and decompressors assign these version numbers.

vendor

Identifies the developer of the compressor that created the compressed image.

temporalQuality

Indicates the degree of temporal compression performed on the image data associated with this description. This field is valid only for sequences. See CompressImage (I-113) for a list of values.

spatialQuality

Indicates the degree of spatial compression performed on the image data associated with this description. This field is valid for sequences and still images. See CompressImage (I-113) for a list of values.

width

Contains the width of the source image, in pixels.

height

Contains the height of the source image, in pixels.

hRes

Contains the horizontal resolution of the source image, in dots per inch.

vRes

Contains the vertical resolution of the source image, in dots per inch.

dataSize

Indicates the size of the compressed image, in bytes. This field is valid only for still images. Set this field to 0 if the size is unknown.

frameCount

Contains the number of frames in the image data associated with this description.

name

Indicates the compression algorithm used to create the compressed data. This algorithm is stored in Pascal string format. It always takes up 32 bytes no matter how long the string is. The 32 bytes consist of 31 bytes plus one length byte. The value of this field should correspond to the compressor type specified by the cType field, as well as to the value of the typeName field in the appropriate compressor name structure returned by GetCodecNameList (I-386). Applications may use the contents of this field to indicate the type of compression used for the associated image.

depth

Contains the pixel depth specified for the compressed image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the depth of color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

clutID

Contains the ID of the color table for the compressed image, or other special values. If this field is set to 0, then a custom color table is defined for the compressed image. You can use the GetImageDescriptionCTable (I-417) function to retrieve the color table. If this field is set to -1, the image does not use a color table.

Discussion

Data in the ImageDescription structure indicates the type of compression that was used, the size of the image when displayed, the resolution at which the image was captured, and so on. In addition, an image description structure may contain additional data in extensions and custom color tables. One image description structure may be associated with one or more compressed frames.

Associated Functions

CompressImage (I-113)

Programming Info

C interface file: ImageCompression.h

See Also

For functions that let you work with custom color tables in ImageDescription structures, see GetImageDescriptionCTable (I-417) and SetImageDescriptionCTable (III-1593). For functions that let you work with ImageDescription structure extensions, see GetImageDescriptionExtension (I-418), AddImageDescriptionExtension (I-25), RemoveImageDescriptionExtension (III-1457), CountImageDescriptionExtensionType (I-143), and GetNextImageDescriptionExtensionType (I-501). For extensions to the ImageDescription structure, see NCLCColorInfoImageDescriptionExtension

ImageRangeRecord

(IV–2321), PixelAspectRatioImageDescriptionExtension (IV–2332), and CleanApertureImageDescriptionExtension (IV–2174).

ImageRangeRecord

Undocumented

```
struct ImageRangeRecord {
    long    imageFlags;
    OSType  fileType;
    long    replacedAtoms;
};
```

imageFlags

Undocumented

fileType

File type to contain the preset group (normally kStandardPresetGroup).

replacedAtoms

Number of atoms at this level replaced by this preset group.

Discussion

Undocumented

Programming Info

C interface file: ImageCodec.h

ImageSubCodecDecompressCapabilities

Returned by an image decompressor component in response to ImageCodecInitialize (II–724).

```
struct ImageSubCodecDecompressCapabilities {
    long    recordSize;
    long    decompressRecordSize;
    Boolean  canAsync;
    UInt8   pad0;
```

recordSize

The size of this structure in bytes.

decompressRecordSize

The size of the ImageSubCodecDecompressRecord (IV–2289) structure that your image decompressor component requires. This structure is used to pass information from ImageCodecBeginBand (II–690) to ImageCodecDrawBand (II–697) and ImageCodecEndBand (II–704).

canAsync

Specifies whether your image decompressor component can perform asynchronous scheduled decompression. This should be TRUE unless your image decompressor component calls functions that cannot be called during interrupt time.

pad0

Unused.

Discussion

The first function call that your image decompressor component receives from the base image decompressor is always a call to ImageCodecInitialize (II–724). In response to this call, your image decompressor component returns an ImageSubCodecDecompressCapabilities structure that specifies its capabilities.

Associated Functions

ImageCodecInitialize (II–724)

Programming Info

C interface file: ImageCodec.h

See Also

See ImageCodecInitialize (II–724), ImageCodecBeginBand (II–690), ImageCodecDrawBand (II–697), and ImageCodecEndBand (II–704).

ImageSubCodecDecompressRecord

Contains information needed for decompressing a frame.

```
struct ImageSubCodecDecompressRecord {
    Ptr          baseAddr;
    long         rowBytes;
    Ptr          codecData;
    ICMProgressProcRecord progressProcRecord;
    ICMDataProcRecord dataProcRecord;
    void *       userDecompressRecord;
    UInt8        frameType;
    UInt8        pad[3];
    long         priv[2];
};
```

ImageSubCodecDecompressRecord

baseAddr

The address of the destination pixel map, which includes adjustment for the offset. Note that if the bit depth of the pixel map is less than 8, your image decompressor component must adjust for the bit offset.

rowBytes

The offset in bytes from one row of the destination pixel map to the next. The value of the rowBytes field must be less than 0x4000.

codecData

A pointer to the data to be decompressed.

progressProcRecord

An ICMProgressProcRecord (IV–2284) structure that specifies a **progress function**. This function reports on the progress of a decompression operation. If there is no progress function, the Image Compression Manager sets the progressProc field in the ICMProgressProcRecord structure to NIL.

dataProcRecord

An ICMDataProcRecord (IV–2279) structure that specifies a data-loading function. If the data to be decompressed is not all in memory, your component can call this function to load more data. If there is no data-loading function, the Image Compression Manager sets the dataProc field in the ICMDataProcRecord structure to NIL, and the entire image must be in memory at the location specified by the codecData field of the ImageSubCodecDecompressRecord structure.

userDecompressRecord

A pointer to storage for the decompression operation. The storage is allocated by the base image decompressor after it calls ImageCodecInitialize (II–724). The size of the storage is determined by the decompressRecordSize field of the ImageSubCodecDecompressCapabilities structure that is returned by ImageCodecInitialize. Your image decompressor component should use this storage to store any additional information needed about the frame in order to decompress it.

frameType

A constant (see below) that indicates the frame type.

pad

Unused.

priv

Private to QuickTime; do not use.

frameType Constants

kCodecFrameTypeUnknown

The frame type is unknown.

kCodecFrameTypeKey

This is a **key frame**.

kCodecFrameTypeDifference

This is a difference frame.

kCodecFrameTypeDroppableDifference

This is a droppable difference frame.

Associated Functions

ImageCodecBeginBand (II–690)

Programming Info

C interface file: ImageCodec.h

InstCompInfo

Contains a list of all atomic instruments supported by an instrument component.

```

struct InstCompInfo {
    long                infoSize;
    Str31               InstrumentLibraryName;
    QAtomContainer      InstrumentLibraryITxt;
    long                GMinstrumentCount;
    GMinstrumentInfoHandle GMinstrumentInfo;
    long                GMdrumCount;
    GMinstrumentInfoHandle GMdrumInfo;
    long                nonGMinstrumentCount;
    nonGMinstrumentInfoHandle nonGMinstrumentInfo;
};

```

infoSize

The size of this structure in bytes.

InstrumentLibraryName

The readable name of this instrument library, up to 31 bytes.

InstrumentLibraryITxt

International text name atoms for instruments.

GMinstrumentCount

The number of General MIDI instruments.

InstKnobList

GMInstrumentInfo

A handle to a list of General MIDI instrument information structures.

GMdrumCount

The number of General MIDI drum kits.

GMdrumInfo

A handle to a list of General MIDI instrument information structures.

nonGMInstrumentCount

The number of non-General MIDI instruments.

nonGMInstrumentInfo

A handle to the list of non-General MIDI instruments.

Associated Functions

InstrumentGetInfo (II-766)

Programming Info

C interface file: QuickTimeMusic.h

InstKnobList

Contains a list of InstKnobRec (IV-2293) structures.

```
struct InstKnobList {  
    BigEndianLong    knobCount;  
    BigEndianLong    knobFlags;  
    InstKnobRec      knob[1];  
};
```

knobCount

The number of InstKnobRec structures in the list.

knobFlags

Constants (see below) that tell what to do if a requested knob is not in the list.

knob

An array of InstKnobRec (IV-2293) structures.

knobFlags Constants

kInstKnobMissingUnknown

If the requested knob is not in the list, do not set its value.

kInstKnobMissingDefault

If the requested knob is not in the list, use its default value.

Programming Info

C interface file: QuickTimeMusic.h

InstKnobRec

Contains information about an instrument knob.

```
struct InstKnobRec {  
    BigEndianLong    number;  
    BigEndianLong    value;  
};
```

number

A knob ID or index. A nonzero value in the high byte indicates that it is an ID. The knob index ranges from 1 to the number of knobs; the ID is an arbitrary number.

value

The value the knob is set to.

Programming Info

C interface file: QuickTimeMusic.h

InstLibDescRec

Contains the name of an instrument library.

```
struct InstLibDescRec {  
    Str31    libIDName;  
};
```

libIDName

The library's readable name.

Programming Info

C interface file: QuickTimeMusic.h

InstrumentAboutInfo

Contains the information that appears in an instrument's About box and is returned by MusicGetInstrumentAboutInfo (II-990).

InstrumentInfoList

```
struct InstrumentAboutInfo {  
    PicHandle    p;  
    Str255       author;  
    Str255       copyright;  
    Str255       other;  
};
```

p
A handle to a graphic for the About box.

author
The author's name.

copyright
The copyright information.

other
Any other textual information.

Associated Functions

MusicGetInstrumentAboutInfo (II-990)

Programming Info

C interface file: QuickTimeMusic.h

See Also

See MusicGetInstrumentAboutInfo (II-990).

InstrumentInfoList

Contains the list of instruments available on a synthesizer.

```
struct InstrumentInfoList {  
    long          recordCount;  
    Handle        toneNames;  
    QTAtomContainer itxtNames;  
    InstrumentInfoRecord info[1];  
};
```

recordCount
The number of InstrumentInfoRecord (IV-2295) structures in the list.

toneNames
A string list of the instrument names as specified in their tone descriptions.

itxtNames
A list of international text names, taken from the name atoms.

info

An array of InstrumentInfoRecord (IV–2295) structures.

Associated Functions

MusicGetInstrumentInfo (II–991)

Programming Info

C interface file: QuickTimeMusic.h

See Also

See InstrumentInfoRecord (IV–2295).

InstrumentInfoRecord

Provides information about non-General MIDI instruments within an instrument component.

```
struct InstrumentInfoRecord {
    long    instrumentNumber;
    long    flags;
    long    toneNameIndex;
    long    itxtNameAtomID;
};
```

instrumentNumber

The number of the instrument within the instrument component. If the ID is 0, the name is a category name.

flags

Unused. Set to 0.

toneNameIndex

The instrument's position in the toneNames index stored in the instrument information list this structure is a part of. The index is a 1-based index.

itxtNameAtomID

The instrument's position in the itxtNames index stored in the instrument information list this structure is a part of.

Programming Info

C interface file: QuickTimeMusic.h

InstSampleDescRec

Contains a description of an audio sample, including sample rate, loop points, and lowest and highest key to play on.

```
struct InstSampleDescRec {
    BigEndianOSType      dataFormat;
    BigEndianShort       numChannels;
    BigEndianShort       sampleSize;
    BigEndianUnsignedFixed sampleRate;
    BigEndianShort       sampleDataID;
    BigEndianLong        offset;
    BigEndianLong        numSamples;
    BigEndianLong        loopType;
    BigEndianLong        loopStart;
    BigEndianLong        loopEnd;
    BigEndianLong        pitchNormal;
    BigEndianLong        pitchLow;
    BigEndianLong        pitchHigh;
};
```

dataFormat

The data format type (see below).

numChannels

The number of channels of data present in the sample.

sampleSize

The size of the sample; 8-bit or 16-bit.

sampleRate

The rate at which to play the sample in unsigned fixed-point 16.16.

sampleDataID

The ID number of a sample data atom that contains the sample audio data.

offset

Reserved. Set to 0.

numSamples

The number of data samples in the sound.

loopType

A constant (see below) indicating the type of loop.

loopStart

Indicates the beginning of the portion of the sample that is looped if the sound is sustained. The position is given in the number of data samples from the start of the sound.

loopEnd

Indicates the end of the portion of the sample that is looped if the sound is sustained. The position is given in the number of data samples from the start of the sound.

pitchNormal

The number of the MIDI note produced if the sample is played at the rate specified in `sampleRate`.

pitchLow

The lowest pitch at which to play the sample. Use for instruments, such as pianos, that have different samples to use for different pitch ranges.

pitchHigh

The highest pitch at which to play the sample. Use for instruments, such as pianos, that have different samples to use for different pitch ranges.

dataFormat Constants

`'twos'`

Signed data.

`'raw '`

Unsigned data.

loopType Constants

`kMusicLoopTypeNormal`

Use a regular loop.

`kMusicLoopTypePalindrome`

Use a back-and-forth loop.

Programming Info

C interface file: `QuickTimeMusic.h`

ITab

Contains a Color QuickDraw inverse table.

```
struct ITab {
    long    iTabSeed;
    short   iTabRes;
    Byte    iTTable[1];
};
```

iTabSeed

A copy of the `ctSeed` field from the source `ColorTable` (IV-2210) structure.

KaraokeAtom

iTabRes

Channel resolution of the inverse table: 3, 4, or 5 bits.

iTTable

Byte-length ColorTable (IV–2210) index values.

Programming Info

C interface file: Quickdraw.h

See Also

See ColorTable (IV–2210). For information about inverse tables, see Chapter 7 of *Advanced Color Imaging on the Mac OS* (listed in the **bibliography**).

KaraokeAtom

Associates highlighted runs of text with movie times.

```
struct KaraokeAtom {  
    long          numEntries;  
    KaraokeRec    karaokeEntries[1];  
};
```

numEntries

The number of entries in karaokeEntries.

karaokeEntries

An array of KaraokeRec (IV–2298) data structures.

Programming Info

C interface file: MoviesFormat.h

KaraokeRec

Associates highlighted text with a movie time value.

```
struct KaraokeRec {  
    TimeValue    timeVal;  
    short        beginHilite;  
    short        endHilite;  
};
```

timeVal

The time location of the highlighted text.

beginHilite

Character number of highlighted text start character.

endHilite

Character number of highlighted text end character.

Programming Info

C interface file: `MoviesFormat.h`

See Also

See `KaraokeAtom` (IV–2298).

KnobDescription

Contains sound parameter values for a single knob.

```
struct KnobDescription {
    Str63    name;
    long     lowValue;
    long     highValue;
    long     defaultValue;
    long     flags;
    long     knobID;
};
```

`name`

The name of the knob.

`lowValue`

The lowest number you can set the knob to.

`highValue`

The highest number you can set the knob to.

`defaultValue`

A value to use for the default. A default instrument is made of all default values.

`flags`

Constants (see below) that provide various items of information about the knob.

`knobID`

A knob ID or index. A nonzero value in the high byte indicates that it is an ID. The knob index ranges from 1 to the number of knobs; the ID is an arbitrary number. Use the knob ID to refer to the knob in preference to the knob index, which may change.

KnobDescription

flags Constants

kKnobReadOnly

The knob value cannot be changed by the user or with a set knob call.

kKnobInterruptUnsafe

Alter this knob only from foreground task time.

kKnobKeyrangeOverride

The knob can be overridden within a single key range (software synthesizer only).

kKnobGroupStart

The knob is first in some logical group of knobs.

kKnobFixedPoint8

Interpret knob numbers as fixed-point 8-bit.

kKnobFixedPoint16

Interpret knob numbers as fixed-point 16-bit.

kKnobTypeNumber

The knob value is a numerical value.

kKnobTypeGroupName

The name of the knob is really a group name for display purposes.

kKnobTypeBoolean

The knob is an on/off knob. If the range of the knob (as specified by the low value and high value in the knob description structure) is greater than one, the knob is a multi-checkbox field.

kKnobTypeNote

The knob value range is equivalent to MIDI keys.

kKnobTypePan

The knob value is the pan setting and is within a range (as specified by the low value and high value in the knob description structure) that goes from left to right.

kKnobTypeInstrument

The knob value is a reference to another instrument number.

kKnobTypeSetting

The knob value is one of several different discrete settings; for example, items on a pop-up menu.

kKnobTypeMilliseconds

The knob value is in milliseconds.

- kKnobTypePercentage
The knob value is a percentage of the range.
- kKnobTypeHertz
The knob value represents frequency.
- kKnobTypeButton
The knob is a momentary trigger push button.

Associated Functions
MusicGetDrumKnobDescription (II–988)

Programming Info
C interface file: QuickTimeMusic.h

LeftOverBlock

Holds the contents of the leftOverSamples field of a CmpSoundHeader (IV–2176) structure.

```
struct LeftOverBlock {
    unsigned long    count;
    SInt8            sampleArea[32];
};
```

- count
The number of bytes in the sampleArea field.
- sampleArea
An array of bytes. This field contains samples that are truncated across invocations of the compression algorithm. The size of each block is defined by a constant (see below).

sampleArea Constant
leftOverBlockSize
Value is 32.

Programming Info
C interface file: Sound.h

LevelMeterInfo

Contains sound level meter readings.

```
struct LevelMeterInfo {
```

LongRangeRecord

```
        short    numChannels;  
        UInt8    leftMeter;  
        UInt8    rightMeter;  
};
```

numChannels

Contains 1 for mono or 2 for stereo source.

leftMeter

Left meter level, 0-255 range.

rightMeter

Right meter level, 0-255 range.

Associated Functions

MediaGetSoundLevelMeterInfo (II-863)

Programming Info

C interface file: Sound.h

LongRangeRecord

Undocumented

```
struct LongRangeRecord {  
    long    minValue;  
    long    maxValue;  
    long    scaleValue;  
    long    precisionDigits;  
};
```

minValue

Undocumented

maxValue

Undocumented

scaleValue

Undocumented

precisionDigits

Undocumented

Discussion

Undocumented

Programming Info

C interface file: ImageCodec.h

MacPolygon

Defines a polygon shape.

```
struct MacPolygon {  
    short    polySize;  
    Rect     polyBBox;  
    Point    polyPoints[1];  
};
```

polySize

The size in bytes of this structure.

polyBBox

The rectangle that bounds the polygon.

polyPoints

An array of points, beginning with the starting point and followed by each successive point to which a line is drawn.

Version Notes

MacPolygon supersedes the Polygon structure to avoid name conflicts in Windows.

Programming Info

C interface file: Quickdraw.h

MacRegion

Defines a two-dimensional region in Macintosh graphics.

```
struct MacRegion {  
    unsigned short    rgnSize;  
    Rect              rgnBBox;  
};
```

rgnSize

The size in bytes of this structure. If the region is rectangular or empty, rgnSize is 10.

rgnBBox

The region's bounding rectangle.

Discussion

This structure can be followed by additional data (in a private format) if the region is not rectangular.

MatrixRecord

Version Notes

MacRegion supersedes the Region structure to avoid name conflicts with Windows.

Programming Info

C interface file: Quickdraw.h

MatrixRecord

Contains a transformation matrix.

```
struct MatrixRecord {  
    Fixed    matrix[3][3];  
};
```

matrix

A 3-by-3 array of matrix values.

Associated Functions

GetMovieMatrix (I-478)

Programming Info

C interface file: ImageCompression.h

MediaEQSpectrumBandsRecord

Provides data for MediaGetSoundEqualizerBands (II-862) and MediaSetSoundEqualizerBands (II-906).

```
struct MediaEQSpectrumBandsRecord {  
    short          count;  
    UnsignedFixedPtr frequency;  
};
```

count

Number of frequencies in this structure.

frequency

Pointer to array of frequencies.

Associated Functions

MediaGetSoundEqualizerBands (II-862)

Programming Info

C interface file: MediaHandlers.h

MediaHeader

Specifies the characteristics of a media.

```
struct MediaHeader {
    long      flags;
    long      creationTime;
    long      modificationTime;
    TimeValue timeScale;
    TimeValue duration;
    short     language;
    short     quality;
};
```

- flags
 One byte of version information followed by three bytes of flags. The flags bytes are not currently used.
- creationTime
 Number of seconds since midnight, January 1, 1904 to the time when the 'mdia' (IV-2560) atom was created.
- modificationTime
 Number of seconds since midnight, January 1, 1904 to the time when the 'mdia' (IV-2560) atom was last changed.
- timeScale
 The number of media time units that pass during each real-time second.
- duration
 The duration of the media in media time units.
- language
 The language code for this media; see “Localization Codes” (IV-2673).
- quality
 Constants (see below) that describe the environment in which this media can be most suitably played.

quality Constants

- 0x00000001
 Specifies a pixel depth of 1 bit per pixel.
- 0x00000010
 Specifies a pixel depth of 2 bits per pixel.
- 0x00000100
 Specifies a pixel depth of 4 bits per pixel.

MediaPacketizerInfo

0x00001000

Specifies a pixel depth of 8 bits per pixel.

0x00010000

Specifies a pixel depth of 16 bits per pixel.

0x00100000

Specifies a pixel depth of 32 bits per pixel.

mediaQualityDraft

Specifies the lowest quality level. Bits 6 and 7 = 00.

mediaQualityNormal

Specifies an acceptable quality level. Bits 6 and 7 = 01.

mediaQualityBetter

Specifies a higher quality level. Bits 6 and 7 = 10.

mediaQualityBest

Specifies the highest quality level. Bits 6 and 7 = 11.

Programming Info

C interface file: MoviesFormat.h

See Also

For the atom structure that contains the `MediaHeader` data structure, see 'mdhd' (IV–2559).

MediaPacketizerInfo

Provides information about a media packetizer.

```
struct MediaPacketizerInfo {
    OSType          mediaType;
    OSType          dataFormat;
    OSType          vendor;
    UInt32          capabilityFlags;
    UInt8           canPackMatrixType;
    UInt8           pad[3];
    long            characteristicCount;
    RTPPayloadCharacteristic characteristic[1];
};
```

mediaType

Media type supported; see “Data References” (IV–2661). 0 means all media types.

dataFormat

Data format supported; see “Codec Identifiers” (IV–2655). 0 means all formats.

vendor

Manufacturer of this packetizer; e.g., 'appl' for Apple.

capabilityFlags

Constants (see below) that indicate the packetizer's ability to handle non-standard track characteristics.

canPackMatrixType

Constant (see below); the packetizer can pack any matrix type up to this level. Set to identityMatrixType for identity matrix (no translation) only.

pad

Unused.

characteristicCount

The number of RTPPayloadCharacteristic structures in the characteristic field.

characteristic

An array of RTPPayloadCharacteristic (IV–2409) structures.

capabilityFlags Constants

kMediaPacketizerCanPackEditRate

The packetizer can pack the edit rate value.

kMediaPacketizerCanPackLayer

The packetizer can pack the layer number.

kMediaPacketizerCanPackVolume

The packetizer can pack the sound volume value.

kMediaPacketizerCanPackBalance

The packetizer can pack the sound balance value.

kMediaPacketizerCanPackGraphicsMode

The packetizer can pack the graphics transfer mode value.

kMediaPacketizerCanPackEmptyEdit

The packetizer can pack the empty edit value.

canPackMatrixType Constants

identityMatrixType

Matrix is identity; value is 0x00.

translateMatrixType

Matrix translates; value is 0x01.

scaleMatrixType

Matrix scales; value is 0x02.

MediaPacketizerRequirements

scaleTranslateMatrixType

Matrix translates and scales; value is 0x03.

linearMatrixType

Matrix is general 2 x 2 type; value is 0x04.

linearTranslateMatrixType

Matrix is general 2 x 2 type and translates; value is 0x05

perspectiveMatrixType

Matrix is general 3 x 3 type; value is 0x06.

Discussion

The last characteristic is followed by the payload name, defined by the RTPPayloadInfo (IV–2409) structure.

Programming Info

C interface file: QTStreamingComponents.h

MediaPacketizerRequirements

Stores the functional requirements for a media packetizer.

```
struct MediaPacketizerRequirements {  
    OSType    mediaType;  
    OSType    dataFormat;  
    UInt32    capabilityFlags;  
    UInt8     canPackMatrixType;  
    UInt8     pad[3];  
};
```

mediaType

Media type required; see “Data References” (IV–2661). 0 means all media types.

dataFormat

Data format required; see “Codec Identifiers” (IV–2655). 0 means all formats.

capabilityFlags

Constants (see below) that indicate the packetizer’s ability to handle non-standard track characteristics.

canPackMatrixType

Constant (see below); the packetizer needs to pack any matrix type up to this level. Set to identityMatrixType for identity matrix (no translation) only.

pad

Unused.

capabilityFlags Constants

kMediaPacketizerCanPackEditRate

The packetizer can pack the edit rate value.

kMediaPacketizerCanPackLayer

The packetizer can pack the layer number.

kMediaPacketizerCanPackVolume

The packetizer can pack the sound volume value.

kMediaPacketizerCanPackBalance

The packetizer can pack the sound balance value.

kMediaPacketizerCanPackGraphicsMode

The packetizer can pack the graphics transfer mode value.

kMediaPacketizerCanPackEmptyEdit

The packetizer can pack the empty edit value.

canPackMatrixType Constants

identityMatrixType

Matrix is identity; value is 0x00.

translateMatrixType

Matrix translates; value is 0x01.

scaleMatrixType

Matrix scales; value is 0x02.

scaleTranslateMatrixType

Matrix translates and scales; value is 0x03.

linearMatrixType

Matrix is general 2 x 2 type; value is 0x04.

linearTranslateMatrixType

Matrix is general 2 x 2 type and translates; value is 0x05

perspectiveMatrixType

Matrix is general 3 x 3 type; value is 0x06.

Associated Functions

QTSFindMediaPacketizer (II-1231)

Programming Info

C interface file: QTStreamingComponents.h

MediaRecord

MediaRecord

Undocumented

```
struct MediaRecord {  
    long    data[1];  
};
```

data

Undocumented

Programming Info

C interface file: Movies.h

MenuInfo

Contains information about a Mac OS menu.

```
struct MenuInfo {  
    short    menuID;  
    short    menuWidth;  
    short    menuHeight;  
    Handle    menuProc;  
    long    enableFlags;  
    Str255    menuData;  
};
```

menuID

A unique number that identifies the menu.

menuWidth

The menu's horizontal dimension in pixels.

menuHeight

The menu's vertical dimension in pixels.

menuProc

A handle to a menu definition function. The Mac OS uses this function to draw the menu.

enableFlags

Bits that represent the enabled status of the menu title and the first 31 items in the menu. A bit value of 1 means that item is enabled. Items after 31 are always enabled.

menuData

The title of the menu.

Discussion

Application software should never directly change the fields of this structure.

Programming Info

C interface file: Menus.h

STR

Data Structures

ModifierTrackGraphicsModeRecord

Contains information that defines the graphics mode setting for a track.

```
struct ModifierTrackGraphicsModeRecord {
    long        graphicsMode;
    RGBColor    opColor;
};
```

graphicsMode

Specifies the graphics mode setting (see below).

opColor

Contains an RGB color structure indicating the opcolor to use with the graphics mode.

graphicsMode Constants

fullScreenHideCursor

If this flag is set, BeginFullScreen hides the cursor. This is useful if you are going to play a QuickTime movie and do not want the cursor to be visible over the movie.

fullScreenAllowEvents

If this flag is set, your application intends to allow other applications to run (by calling WaitNextEvent to grant them processing time). In this case, BeginFullScreen does not change the monitor resolution, because other applications might depend on the current resolution.

fullScreenDontChangeMenuBar

If this flag is set, BeginFullScreen does not hide the menu bar. This is useful if you want to change the resolution of the monitor but still need to allow the user to access the menu bar.

fullScreenPreflightSize

If this flag is set, BeginFullScreen doesn't change any monitor settings, but returns the actual height and width that it would use if this bit were not set. This

ModRef

allows applications to test for the availability of a monitor setting without having to switch to it.

Discussion

Data in this structure indicates the graphics mode setting and the RGB opcolor that are used with certain graphics modes. Data sent to the `kTrackModifierTypeGraphicsMode` track modifier input type should be in this structure.

Programming Info

C interface file: `Movies.h`

ModRef

Provides data for a `SndListResource` (IV–2447) structure.

```
struct ModRef {
    unsigned short    modNumber;
    long              modInit;
};
```

`modNumber`

Undocumented

`modInit`

Undocumented

Programming Info

C interface file: `Sound.h`

MotionJPEGApp1Marker

Undocumented

```
struct MotionJPEGApp1Marker {
    long    unused;
    long    tag;
    long    fieldSize;
    long    paddedFieldSize;
    long    offsetToNextField;
    long    qTableOffset;
    long    huffmanTableOffset;
    long    sofOffset;
    long    sosOffset;
    long    soiOffset;
```

```
};

unused
    Undocumented
tag
    Undocumented
fieldSize
    Undocumented
paddedFieldSize
    Undocumented
offsetToNextField
    Undocumented
qTableOffset
    Undocumented
huffmanTableOffset
    Undocumented
sofOffset
    Undocumented
sosOffset
    Undocumented
soiOffset
    Undocumented
```

Programming Info

C interface file: ImageCodec.h

MovieEditStateRecord

```
Undocumented

struct MovieEditStateRecord {
    long    data[1];
};

data
    Undocumented
```

Programming Info

C interface file: Movies.h

MovieExportGetDataParams

Parameter block used by `MovieExportGetDataProc` (III–2101).

```
struct MovieExportGetDataParams {  
    long                recordSize;  
    long                trackID;  
    TimeScale           sourceTimeScale;  
    TimeValue           requestedTime;  
    TimeValue           actualTime;  
    Ptr                 dataPtr;  
    long                dataSize;  
    SampleDescriptionHandle desc;  
    OSType              descType;  
    long                descSeed;  
    long                requestedSampleCount;  
    long                actualSampleCount;  
    TimeValue           durationPerSample;  
    long                sampleFlags;  
};
```

`recordSize`

Contains the total size in bytes of the `MovieExportGetDataParams` structure. This is provided to allow for additional parameters to be added safely in the future.

`trackID`

Specifies the data source. The `trackID` is returned when the data source is added by calling `MovieExportAddDataSource` (II–919).

`sourceTimeScale`

Specifies the time scale for this data source. This value is the same time scale that is passed to `MovieExportAddDataSource` (II–919).

`requestedTime`

Specifies the time of the media requested by the exporter. The time scale is the same one specified when adding a data source with `MovieExportAddDataSource` (II–919).

`actualTime`

Specifies the time actually referred to by the returned media data. This value is provided by `MovieExportGetDataProc` (III–2101), and is usually the same as `requestedTime`.

`dataPtr`

Contains a 32-bit pointer to the media data.

`dataSize`

Specifies the size in bytes of the data pointed to by `dataPtr`.

desc

Contains a `SampleDescriptionHandle` describing the format of the data pointed to by `dataPtr`. For video data, this is an `ImageDescriptionHandle`. For sound data, this is a `SoundDescriptionHandle`.

descType

Specifies the type of `SampleDescriptionHandle`; see “Component Types” (IV–2621). For example, if `SampleDescriptionHandle` is `ImageDescriptionHandle`, `descType` is set to `VideoMediaType`.

descSeed

Specifies which `SampleDescriptionHandle` represented by the current value of `desc`. Some data sources contain different kinds of data at different times. For example, a video data source may contain both JPEG and uncompressed raw data. Whenever the data source switches from one type of data to another, change `descSeed` to notify the exporter. In the case of an export operation that is providing its source data from a QuickTime movie track, `descSeed` is equal to the sample description index of the sample being returned.

requestedSampleCount

Specifies the number of samples the exporter can work with. The function can return more or fewer samples than requested. For video, this value is always 1.

actualSampleCount

Specifies the number of samples actually returned. Your `MovieExportGetDataProc` (III–2101) function must fill in this field.

durationPerSample

Specifies the duration of every sample returned. For sound data, `durationPerSample` always contains 1. For video data, `durationPerSample` contains the duration of the returned sample, expressed in the time scale defined when the data source was created.

sampleFlags

Contains flags (see below) for the returned samples. The only defined flag is `mediaSampleNotSync`. Your `MovieExportGetDataProc` (III–2101) function must fill in this field.

sampleFlags Constants

`mediaSampleNotSync`

Returned for frame-differenced video sample data.

Discussion

Your `MovieExportGetDataProc` (III–2101) function is passed to `MovieExportAddDataSource` to define a new data source for an export operation. The function is used by the exporting application to request source media data to be

MovieHeader

used in the export operation. For example, in a video export operation, frames of video data (either compressed or uncompressed) are provided. In a sound export operation, buffers of audio (either compressed or uncompressed) are provided.

The data pointed to by `dataPtr` must remain valid until the next call to your `MovieExportGetDataProc` function. Your `MovieExportGetDataProc` function is responsible for allocating and disposing of the memory associated with this data pointer.

Associated Functions

`MovieExportGetDataProc` (III–2101)

Programming Info

C interface file: `QuickTimeComponents.h`

MovieHeader

Specifies the top-level characteristics of a movie.

```
struct MovieHeader {
    long          flags;
    long          creationTime;
    long          modificationTime;
    TimeValue     timeScale;
    TimeValue     duration;
    Fixed         preferredRate;
    short         preferredVolume;
    short         reserved1;
    long          preferredLong1;
    long          preferredLong2;
    MatrixRecord  matrix;
    TimeValue     previewTime;
    TimeValue     previewDuration;
    TimeValue     posterTime;
    TimeValue     selectionTime;
    TimeValue     selectionDuration;
    TimeValue     currentTime;
    long          nextTrackID;
};
```

flags

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

creationTime

Number of seconds since midnight, January 1, 1904 to the time when the 'moov' (IV-2566) atom was created.

modificationTime

Number of seconds since midnight, January 1, 1904 to the time when the 'moov' (IV-2566) atom was last changed.

timeScale

The number of movie time units that pass during each real-time second.

duration

The duration of the movie in movie time units.

preferredRate

The rate at which to play this movie, ranging from -1.0 to $+1.0$. A value of $+1.0$ indicates normal forward play. Negative values indicate that the movie will play backwards.

preferredVolume

The sound volume at which to play this movie, ranging from -1.0 to $+1.0$. The high-order 8 bits contain the integer part; the low-order 8 bits contain the fractional part. A value of $+1.0$ constitutes the maximum volume of the user's computer. Negative values are silent but retain the magnitude of the volume setting.

reserved1

Reserved; set to 0.

preferredLong1

Reserved; set to 0.

preferredLong2

Reserved; set to 0.

matrix

A MatrixRecord (IV-2304) structure that contains the graphic transformation matrix for this movie.

previewTime

The time value in movie time units at which the **preview** begins.

previewDuration

The duration of the movie **preview** in movie time scale units.

posterTime

The time value in movie time units at which the **poster** is located.

selectionTime

The time value in movie time units at which the current selection begins.

MovieMediaTimeRecord

selectionDuration

The duration of the current selection in movie time scale units.

currentTime

The current time value in movie time units.

nextTrackID

The next value to use for a track ID number in this movie.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For the atom structure that normally contains the `MovieHeader` structure, see 'mvhd' (IV-2567).

MovieMediaTimeRecord

Undocumented

```
struct MovieMediaTimeRecord {  
    wide      time;  
    TimeScale scale;  
};
```

time

The media time.

scale

The time scale.

Programming Info

C interface file: `Movies.h`

MovieRecord

Undocumented

```
struct MovieRecord {  
    long    data[1];  
};
```

data

Undocumented

Programming InfoC interface file: `Movies.h`**MoviesUserData**

Stores user data in movie and track directories.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

size

Size in bytes of the user data.

udType

Type of the user data.

data

User data.

Programming InfoC interface file: `MoviesFormat.h`**See Also**For the atom type that contains `MoviesUserData` structures, see 'udta' (IV–2607).**MusicDescription**

Undocumented

```
struct MusicDescription {
    long          descSize;
    long          dataFormat;
    long          resvd1;
    short         resvd2;
    short         dataRefIndex;
    long          musicFlags;
    unsigned long headerData[1];
};
```

descSize

Undocumented

MusicMIDIPacket

dataFormat
 Contains 'musi'.
resvd1
 Reserved.
resvd2
 Reserved.
dataRefIndex
 Undocumented
musicFlags
 Undocumented
headerData
 Undocumented

Programming Info

C interface file: Movies.h

MusicMIDIPacket

Describes MIDI data passed by note allocation calls.

```
struct MusicMIDIPacket {  
    unsigned short    length;  
    unsigned long     reserved;  
    UInt8             data[249];  
};
```

length
 The length of the data in the packet.

reserved
 Contains 0, or one of the music packet status constants (see below).

data
 MIDI data.

reserved Constants

kMusicPacketPortLost
 The application has lost the default input port.

kMusicPacketPortFound
 The application has retrieved the input port from the previous owner.

kMusicPacketTimeGap
The last byte of the packet specifies how long (in milliseconds) to keep the MIDI line silent after sending the packet.

Associated Functions
MusicDerivedMIDISend (II-975)

Programming Info
C interface file: QuickTimeMusic.h

NCLCColorInfoImageDescriptionExtension

Video color info extension to the ImageDescription (IV-2285) structure.

```
struct NCLCColorInfoImageDescriptionExtension {
    OSType      colorParamType;
    UInt16      primaries;
    UInt16      transferFunction;
    UInt16      matrix;
};

colorParamType
    Value is 'nclc'.

primaries
    CIE 1931 XY chromaticity coordinates.

transferFunction
    Nonlinear transfer function from RGB to ErEgEb.

matrix
    Matrix from ErEgEb to EyEcbEcr.
```

Programming Info
C interface file: ImageCodec.h

See Also
See ImageDescription (IV-2285).

nonGMInstrumentInfo

Holds an array of nonGMInstrumentInfoRecord (IV-2322) structures.

```
struct nonGMInstrumentInfo {
    long                      recordCount;
```

nonGMInstrumentInfoRecord

```
    Handle                toneNames;  
    QTAtomContainer       itxtNames;  
    nonGMInstrumentInfoRecord  instInfo[1];  
};
```

recordCount
 Number of records in instInfo.

toneNames
 Name from tone description.

itxtNames
 International text name atoms for instruments.

instInfo
 An array of nonGMInstrumentInfoRecord (IV–2322) structures.

Programming Info

C interface file: QuickTimeMusic.h

nonGMInstrumentInfoRecord

Provides information about a non-General MIDI instrument within a QTMA instrument component.

```
struct nonGMInstrumentInfoRecord {  
    long    cmpInstID;  
    long    flags;  
    long    toneNameIndex;  
    long    itxtNameAtomID;  
};
```

cmpInstID
 The number of the instrument within the instrument component. If this is 0, the name is a category name.

flags
 Not used.

toneNameIndex
 The instrument's position in the toneNames index stored in the InstrumentInfoList (IV–2294) structure this structure is a part of. The index is a one-based index.

`itxtNameAtomID`

The instrument's position in the `itxtNames` index stored in the `InstrumentInfoList` (IV–2294) structure this structure is a part of.

Programming Info

C interface file: `QuickTimeMusic.h`

See Also

See `nonGMINstrumentInfo` (IV–2321), `InstrumentInfoList` (IV–2294).

NoteRequest

Provides complete information for allocating a note channel.

```
struct NoteRequest {
    NoteRequestInfo  info;
    ToneDescription  tone;
};
```

`info`

A `NoteRequestInfo` (IV–2323) structure.

`tone`

A `ToneDescription` (IV–2487) structure.

Associated Functions

`NAGetNoteRequest` (II–1027)

Programming Info

C interface file: `QuickTimeMusic.h`

NoteRequestInfo

Contains information for allocating a note channel in addition to that included in a `ToneDescription` (IV–2487) structure.

```
struct NoteRequestInfo {
    UInt8          flags;
    UInt8          reserved;
    BigEndianShort  polyphony;
    BigEndianFixed  typicalPolyphony;
};
```

NumVersion

flags

Constants (see below) that specify what to do if the exact instrument requested in a `ToneDescription` (IV–2487) structure is not found.

reserved

Reserved. Set to 0.

polyphony

Maximum number of voices.

typicalPolyphony

Hint for level mixing.

flags Constants

`kNoteRequestNoGM`

Don't use a General MIDI synthesizer.

`kNoteRequestNoSynthType`

Don't use another synthesizer of the same type but with a different name.

Programming Info

C interface file: `QuickTimeMusic.h`

NumVersion

Contains software version and release information in **little-endian** numeric format.

```
struct NumVersion {
    UInt8    nonRelRev;
    UInt8    stage;
    UInt8    minorAndBugRev;
    UInt8    majorRev;
};
```

`nonRelRev`

The revision level of a prerelease version.

`stage`

The development stage. A constant (see below) that specifies pre-alpha, alpha, beta, or final release.

`minorAndBugRev`

The minor revision level. This field is a signed byte in binary-coded decimal format.

majorRev

The major revision level. This field is a signed byte in binary-coded decimal format.

stage Constants

developStage

Prealpha release; value is 20.

alphaStage

Alpha release; value is 40.

betaStage

Beta release; value is 60.

finalStage

Final release; value is 80.

Discussion

Version information is stored in NumVersion in packed BCD. This NumVersion structure is the same as the first four fields of a Mac OS **resource** of type 'vers', but is accessible in **little-endian** format. For **big-endian** format, reverse the order of the fields. For example, version 4.2.1a3 would be stored as 0x04214003.

Programming Info

C interface file: MacTypes.h

OfflineSampleType

Undocumented

```
struct OfflineSampleType {
    unsigned long    numChannels;
    UnsignedFixed    sampleRate;
    unsigned short   sampleSize;
};
```

numChannels

The number of channels; mono = 1, stereo = 2.

sampleRate

The sample rate in Apple Fixed-point representation.

sampleSize

The number of bits in each sample.

Programming Info

C interface file: QuickTimeMusic.h

OpenCPicParams

OpenCPicParams

Specifies resolutions when creating images.

```
struct OpenCPicParams {  
    Rect    srcRect;  
    Fixed   hRes;  
    Fixed   vRes;  
    short   version;  
    short   reserved1;  
    long    reserved2;  
};
```

srcRect

The optimal bounding rectangle for the resolution indicated by the hRes and vRes fields. To display a picture at a resolution other than that specified in the the hRes and vRes fields, your application should compute an appropriate destination rectangle by scaling the image's width and height by the destination resolution divided by the source resolution.

hRes

The best horizontal resolution for the picture. A value of 0x0048000 specifies a horizontal resolution of 72 dpi.

vRes

The best vertical resolution for the picture. A value of 0x0048000 specifies a vertical resolution of 72 dpi.

version

Always set this field to -2.

reserved1

Reserved; set to 0.

reserved2

Reserved; set to 0.

Associated Functions

GetPictureFileHeader (I-504)

Programming Info

C interface file: Quickdraw.h

ParameterAlternateDataEntry

Provides entries for the ParameterAlternateDataType (IV-2327) structure.

```
struct ParameterAlternateDataEntry {
    OSType      dataType;
    QTAtomType  alternateAtom;
};
```

dataType

The data type in which this item is stored.

alternateAtom

The atom type in which to store it.

Programming Info

C interface file: ImageCodec.h

See Also

See ParameterAlternateDataType (IV–2327).

ParameterAlternateDataType

Undocumented

```
struct ParameterAlternateDataType {
    long          numEntries;
    ParameterAlternateDataEntry  entries[1];
};
```

numEntries

The number of entries in the entries field.

entries

An array of ParameterAlternateDataEntry (IV–2326) structures.

Programming Info

C interface file: ImageCodec.h

ParameterAtomTypeAndID

Undocumented

```
struct ParameterAtomTypeAndID {
    QTAtomType  atomType;
    QTAtomID    atomID;
    long        atomFlags;
    Str255      atomName;
};
```

ParameterDataBehavior

atomType

Type of the atom this data comes from or goes into.

atomID

ID of the atom this data comes from or goes into.

atomFlags

Constants (see below) that define options for this atom, or 0 for no options.

atomName

Name of this value type.

atomFlags Constants

'none'

Atom type for no data gotten or set.

0x00000001

The atom can never be interpolated.

0x00000002

The atom can be interpolated, but it is an advanced user operation.

0x00000004

More than one value of the atom can exist, with ascending IDs (for example, lists of colors).

Programming Info

C interface file: ImageCodec.h

ParameterDataBehavior

Undocumented

```
struct ParameterDataBehavior {
    OSType    behaviorType;
    long      behaviorFlags;
    union     u;
};
```

behaviorType

The type of the behavior; see “Parameter Behaviors” (IV–2685).

behaviorFlags

Flags (see below) that define behavior options, or 0 if there are no options.

u

Union containing ControlBehaviors (IV–2224) structures.

behaviorFlags Constants`kGraphicsFlagsGray`

Draw lines and groups in gray.

`kGroupAlignText`

The edit text items in the group have the same size.

`kGroupSurroundBox`

The group should be surrounded with a box.

`kGroupMatrix`

A side-by-side arrangement of the group is okay.

`kGroupNoName`

The name of the group should not be displayed above the box.

`kDisableWhenNotEqual`With radio buttons, checkboxes, or other controls, *Undocumented*.`kDisableWhenEqual`With radio buttons, checkboxes, or other controls, *Undocumented*.`kDisableWhenLessThan`With radio buttons, checkboxes, or other controls, *Undocumented*.`kDisableWhenGreaterThan`With radio buttons, checkboxes, or other controls, *Undocumented*.`kPopupStoreAsString`With pop-up menus, *Undocumented*.**Programming Info**C interface file: `ImageCodec.h`**ParameterDataType**

Undocumented

```
struct ParameterDataType {
    OSType    dataType;
};
```

`dataType`

Type of data for this item.

Programming InfoC interface file: `ImageCodec.h`

ParameterDataUsage

ParameterDataUsage

Undocumented

```
struct ParameterDataUsage {  
    OSType    usageType;  
};
```

usageType

Higher-level purpose of the data or group.

Programming Info

C interface file: ImageCodec.h

ParameterDependancyRecord

Undocumented

```
struct ParameterDependancyRecord {  
    long      dependCount;  
    OSType    depends[1];  
};
```

dependCount

Number of items in the depends field.

depends

Undocumented

Programming Info

C interface file: ImageCodec.h

Pattern

A 64-bit image that defines a repeating design or tone.

```
struct Pattern {  
    UInt8    pat[8];  
};
```

pat

A row of the pattern.

Discussion

A pattern is an 8-by-8 bit square. When it is drawn, adjacent squares are aligned to form a continuous design.

Programming Info

C interface file: Quickdraw.h

Picture

Contains a Mac OS Picture.

```
struct Picture {  
    short    picSize;  
    Rect     picFrame;  
};
```

picSize

The size of the rest of this record for a version 1 picture. The information in this field is useful only for version 1 pictures, which cannot exceed 32 KB in size.

Version 2 and extended version 2 pictures can be much larger than the 32 KB limit imposed by the 2-byte `picSize` field.

picFrame

The bounding rectangle for the picture defined in the rest of this record. This rectangle is used to scale the picture if you draw it into a destination rectangle of a different size.

Discussion

Mac OS Pictures are sequences of saved drawing commands. They make it easier for your application to draw complex images defined in other applications, and for other applications to display images created with your application.

Version Notes

There are three versions of the `Picture` structure. Version 1 format supports only black-and-white drawing operations at 72 dpi. Version 2 format can be used when the current graphics port is a color graphics port. Pictures created in this format support color drawing operations at 72 dpi. The extended version 2 format permits your application to specify resolutions for pictures in color or black and white. To maintain compatibility with the version 1 picture format, the `picSize` field was not changed for the version 2 picture or extended version 2 formats.

Programming Info

C interface file: Quickdraw.h

PixelAspectRatioImageDescriptionExtension

See Also

For information about Mac OS Pictures, see *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

PixelAspectRatioImageDescriptionExtension

Adds spacing information to an ImageDescription (IV–2285) structure.

```
struct PixelAspectRatioImageDescriptionExtension {
    UInt32    hSpacing;
    UInt32    vSpacing;
};
```

hSpacing

Horizontal spacing.

vSpacing

Vertical spacing.

Programming Info

C interface file: ImageCodec.h

See Also

See ImageDescription (IV–2285).

PixMap

Contains information about the dimensions and contents of a pixel image, as well as its storage format, depth, resolution, and color usage.

```
struct PixMap {
    Ptr        baseAddr;
    short      rowBytes;
    Rect       bounds;
    short      pmVersion;
    short      packType;
    long       packSize;
    Fixed      hRes;
    Fixed      vRes;
    short      pixelType;
    short      pixelSize;
    short      cmpCount;
    short      cmpSize;
    OSType     pixelFormat;
    CTabHandle pmTable;
```

```

    void *      pmExt;
};

```

baseAddr

For an onscreen pixel image, a pointer to the first byte of the image. For optimal performance, this should be a multiple of 4. The `baseAddr` field of the `PixMap` record for an offscreen graphics world contains a handle instead of a pointer. Your application should never directly access the `baseAddr` field of the `PixMap` record for an offscreen graphics world.

rowBytes

The offset in bytes from one row of the image to the next. The value must be even, less than 0x4000, and for best performance it should be a multiple of 4. The high 2 bits of `rowBytes` are used as flags. If bit 15 = 1, the data structure pointed to is a `PixMap` structure; otherwise it is a `BitMap` (IV-2164) structure.

bounds

The boundary rectangle, which links the local coordinate system of a graphics port to QuickDraw's global coordinate system and defines the area of the bit image into which QuickDraw can draw. By default, the boundary rectangle is the entire main screen. Do not use the value of this field to determine the size of the screen; instead use the value of the `gdRect` field of the `GDevice` (IV-2264) structure for the screen.

pmVersion

The version number of Color QuickDraw that created this `PixMap` structure. The value of `pmVersion` is normally 0. If `pmVersion` is 4, Color QuickDraw treats the `PixMap` record's `baseAddr` field as **32-bit clean**. All other flags are private. Most applications never need to set this field

packType

The packing algorithm used to compress image data. Color QuickDraw currently supports a `packType` of 0, which means no packing, and values of 1 to 4 for packing direct pixels.

packSize

The size of the packed image in bytes. When the `packType` field contains the value 0, this field is always set to 0.

hRes

The horizontal resolution of the pixel image in pixels per inch. By default, this value is 0x00480000 (for 72 pixels per inch).

vRes

The vertical resolution of the pixel image in pixels per inch. By default, this value is 0x00480000 (for 72 pixels per inch).

PixMap

pixelType

The storage format for a pixel image. Indexed pixels are indicated by a value of 0. Direct pixels are specified by a value of `RGBDirect`, or 16. In the `PixMap` record of the `GDevice` (IV–2264) structure for a direct device, this field is set to `RGBDirect` when the screen depth is set.

pixelSize

The number of bits used to represent a pixel. Indexed pixels can have sizes of 1, 2, 4, and 8 bits; direct pixel sizes are 16 and 32 bits.

cmpCount

The number of components used to represent a color for a pixel. With indexed pixels, each pixel is a single value representing an index in a color table, and therefore this field contains the value 1; the index is the single component. With direct pixels, each pixel contains three components (one integer each for the intensities of red, green, and blue) so this field contains the value 3.

cmpSize

The size in bits of each component for a pixel. Color QuickDraw expects that the sizes of all components are the same, and that the value of the `cmpCount` field multiplied by the value of the `cmpSize` field is less than or equal to the value in the `pixelSize` field.

For an indexed pixel value, which has only one component, the value of the `cmpSize` field is the same as the value of the `pixelSize` field; that is, 1, 2, 4, or 8. For direct pixels there are two additional possibilities. A 16-bit pixel, which has three components, has a `cmpSize` value of 5; this leaves an unused high-order bit, which Color QuickDraw sets to 0. A 32-bit pixel, which has three components (red, green, and blue), has a `cmpSize` value of 8; this leaves an unused high-order byte, which Color QuickDraw sets to 0.

If presented with a 32-bit image (for example, in the `CopyBits` procedure) Color QuickDraw passes whatever bits are there, and it does not set the high byte to 0. Generally, therefore, your application should clear the memory for the image to 0 before creating a 16-bit or 32-bit image.

planeBytes

The offset in bytes from one drawing plane to the next. This field is set to 0.

pmTable

A handle to a `ColorTable` (IV–2210) structure for the colors in this pixel map.

pmReserved

Reserved. This field must be set to 0 for future compatibility.

pixelFormat

The way the pixels are arranged; see “Pixel Formats” (IV–2686).

pmTable

Color map for this structure.

pmExt

Handle to a `PixelFormatExtension` (IV–2335) structure. Set to `NIL` if there is no extension.

Discussion

The pixel map for a window’s color graphics port always consists of the pixel depth, color table, and boundary rectangle of the main screen, even if the window is created on or moved to an entirely different screen.

Version Notes

Earlier versions of this structure were different in the last three fields; see the C interface file for details.

Programming Info

C interface file: `Quickdraw.h`

See Also

See `PixelFormatExtension` (IV–2335).

PixelFormatExtension

Extends the `PixelFormat` (IV–2332) structure.

```
struct PixelFormatExtension {
    long          extSize;
    unsigned long pmBits;
    void *        pmGD;
    long          pmSeed;
    unsigned long reserved0;
    unsigned long reserved1;
    unsigned long reserved2;
    long          longRowBytes;
};
```

extSize

The size of this structure.

pmBits

The `PixelFormat` attributes bitfield.

PixPat

pmGD

A handle to a GDevice (IV–2264) structure.

pmSeed

If this structure changes, this number will change.

reserved0

Reserved for future use.

reserved1

Reserved for future use.

reserved2

Reserved for future use.

longRowBytes

Field used when the value of the rowBytes field of PixMap (IV–2332) is greater than 16382.

Programming Info

C interface file: Quickdraw.h

PixPat

Stores a pixel pattern.

```
struct PixPat {
    short          patType;
    PixMapHandle   patMap;
    Handle         patData;
    Handle         patXData;
    short          patXValid;
    Handle         patXMap;
    Pattern        pat1Data;
};
```

patType

The pattern's type. The value 0 specifies a basic QuickDraw bit pattern, the value 1 specifies a full-color pixel pattern, and the value 2 specifies an RGB pattern.

patMap

A handle to a PixMap (IV–2332) structure that describes the pattern's pixel image. The PixMap structure can contain indexed or direct pixels.

patData

A handle to the pattern's pixel image.

patXData

A handle to an expanded pixel image used internally by Color QuickDraw.

patXValid

A flag that, when set to -1 , invalidates the expanded data.

patXMap

Reserved.

pat1Data

A bit pattern (typically 50 percent gray) to be used when the pattern defined by this `PixPat` structure is drawn into a `GrafPort` record.

Programming Info

C interface file: `Quickdraw.h`

See Also

See *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

PlanarComponentInfo

Describes a part of a planar pixel map.

```
struct PlanarComponentInfo {
    SInt32    offset;
    UInt32    rowBytes;
};
```

offset

The offset to this part.

rowBytes

The offset in bytes from one row of the image to the next.

Programming Info

C interface file: `ImageCodec.h`

See Also

See the `PlanarPixMapInfo` (IV-2337) structure.

PlanarPixMapInfo

Holds an array of `PlanarComponentInfo` (IV-2337) structures.

```
struct PlanarPixMapInfo {
    PlanarComponentInfo    componentInfo[1];
};
```

PlanarPixmapInfoSorensonYUV9

```
};
```

```
componentInfo
```

An array of PlanarComponentInfo (IV–2337) structures.

Programming Info

C interface file: ImageCodec.h

PlanarPixmapInfoSorensonYUV9

Undocumented

```
struct PlanarPixmapInfoSorensonYUV9 {  
    PlanarComponentInfo    componentInfoY;  
    PlanarComponentInfo    componentInfoU;  
    PlanarComponentInfo    componentInfoV;  
};
```

```
componentInfoY
```

Undocumented

```
componentInfoU
```

Undocumented

```
componentInfoV
```

Undocumented

Programming Info

C interface file: ImageCodec.h

PlanarPixmapInfoYUV420

Undocumented

```
struct PlanarPixmapInfoYUV420 {  
    PlanarComponentInfo    componentInfoY;  
    PlanarComponentInfo    componentInfoCb;  
    PlanarComponentInfo    componentInfoCr;  
};
```

```
componentInfoY
```

Undocumented

componentInfoCb
Undocumented
 componentInfoCr
Undocumented

Programming Info

C interface file: ImageCodec.h

STR

Data Structures

Point

Defines the position of a point.

```
struct Point {
    short    v;
    short    h;
};
```

v

The vertical coordinate of the point.

h

The horizontal coordinate of the point.

Programming Info

C interface file: MacTypes.h

PointerDataRefRecord

References data in memory without using a handle.

```
struct PointerDataRefRecord {
    void    *data;
    Size     dataLength;
};
```

data

A pointer to some data.

dataLength

The length of the data in bytes.

PreviewResourceRecord

Discussion

This structure supports references to data in memory, but does not require that data reside within a block allocated by the Mac OS Memory Manager. Here is an example of its use:

```
ComponentInstance  dataHandler;
PointerDataRef     dataref =
    (PointerDataRef)NewHandle(sizeof(PointerDataRefRecord));

(**dataref).data = myPointerToSomeMedia;
(**dataref).dataLength = theLengthOfTheMedia;

osstat = OpenADataHandler(dataRef, PointerDataHandlerSubType, NIL,
    (OSType)0, NIL, kDataHCanRead, &dataHandler);
... etc.
```

Version Notes

Introduced in QuickTime 5.

Programming Info

C interface file: QuickTimeComponents.h

PreviewResourceRecord

Contains information about a **preview**.

```
struct PreviewResourceRecord {
    unsigned long    modDate;
    short            version;
    OSType            resType;
    short             resID;
};
```

modDate

Contains the modification time (in the standard Macintosh format of seconds since midnight, January 1, 1904) of the file for which the **preview** was created. This parameter allows you to find out if the preview is out of date with the contents of the file.

version

Contains the version number of the **preview resource**. The low bit of the version is a flag for preview components that only reference their data. If the bit is set, it indicates that the resource identified in the preview resource is not

owned by the preview component, but is part of the file. It is not removed when the preview is updated or removed, as it would if the version number were 0.

resType

Identifies the type of the **preview** component used to display the preview data and the type of the atom containing the preview data.

resID

Contains the index (1-based) of the atom to be used. For example, a resType of PICT and a resID of 2 tells QuickTime to use the second PICT atom in the file for the **preview** data.

Programming Info

C interface file: ImageCompression.h

PublicHandlerInfo

Contains data for a 'hdlr' (IV-2540) atom.

```
struct PublicHandlerInfo {
    long    flags;
    long    componentType;
    long    componentSubType;
    long    componentManufacturer;
    long    componentFlags;
    long    componentFlagsMask;
    char    componentName[1];
};
```

flags

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

componentType

A four-character code that identifies the type of component. See “Codec Identifiers” (IV-2655). All components of a particular type support a common set of interface routines. Your application uses this field to search for components of a given type.

componentSubType

A four-character code that identifies the subtype of the component. Different subtypes of a component type may support additional features or provide interfaces that extend beyond the standard routines for a given component type. For example, the subtype of an image compressor component indicates the compression algorithm employed by the compressor. Your application can

QDProcs

use the `componentSubType` field to perform a more specific lookup operation than is possible using only the `componentType` field. Set this parameter to 0 to select a component with any subtype value.

`componentManufacturer`

A four-character code that identifies the manufacturer of the component. This field allows for further differentiation between individual components. For example, components made by a specific manufacturer may support an extended feature set. Components provided by Apple have a manufacturer value of 'appl'. Your application can use this field to find components from a certain manufacturer. A value of 0 operates as a wildcard

`componentFlags`

A 32-bit field that contains flags describing your component's capabilities. The high-order 8 bits are defined by the Component Manager. You should set these bits to 0. The low-order 24 bits are specific to each component type. These flags can be used to indicate the presence of features or capabilities in a given component.

`componentFlagsMask`

A 32-bit field that indicates which flags in the `componentFlags` field are relevant to a particular component search operation. For each flag in the `componentFlags` field that is to be considered as a search criterion, your application should set the corresponding bit in this field to 1. To ignore a flag, set the bit to 0.

`componentName`

The component's name.

Programming Info

C interface file: `MoviesFormat.h`

See Also

See 'hdlr' (IV-2540).

QDProcs

Stores **Universal Procedure Pointers** to low-level drawing routines.

```
struct QDProcs {
    QDTextUPP    textProc;
    QDLineUPP    lineProc;
    QDRectUPP    rectProc;
    QDRRectUPP   rRectProc;
    QDOvalUPP    ovalProc;
    QDArcUPP     arcProc;
    QDPolyUPP    polyProc;
```

```

QDRgnUPP      rgnProc;
QDBitsUPP     bitsProc;
QDCommentUPP  commentProc;
QDTxMeasUPP   txMeasProc;
QDGetPicUPP   getPicProc;
QDPutPicUPP   putPicProc;
};

```

textProc

A **Universal Procedure Pointer** to the low-level routine that draws text. The standard QuickDraw routine is the StdText procedure.

lineProc

A **Universal Procedure Pointer** to the low-level routine that draws lines. The standard QuickDraw routine is the StdLine procedure.

rectProc

A **Universal Procedure Pointer** to the low-level routine that draws rectangles. The standard QuickDraw routine is the StdRect procedure.

rRectProc

A **Universal Procedure Pointer** to the low-level routine that draws rounded rectangles. The standard QuickDraw routine is the StdRRect procedure.

ovalProc

A **Universal Procedure Pointer** to the low-level routine that draws ovals. The standard QuickDraw routine is the StdOval procedure.

arcProc

A **Universal Procedure Pointer** to the low-level routine that draws arcs. The standard QuickDraw routine is the StdArc procedure.

polyProc

A **Universal Procedure Pointer** to the low-level routine that draws polygons. The standard QuickDraw routine is the StdPoly procedure.

rgnProc

A **Universal Procedure Pointer** to the low-level routine that draws regions. The standard QuickDraw routine is the StdRgn procedure.

bitsProc

A **Universal Procedure Pointer** to the low-level routine that copies bitmaps. The standard QuickDraw routine is the StdBits procedure.

commentProc

A **Universal Procedure Pointer** to the low-level routine for processing a picture comment. The standard QuickDraw routine is the StdComment procedure.

QElem

txMeasProc

A **Universal Procedure Pointer** to the low-level routine for measuring text width. The standard QuickDraw routine is the StdTxMeas function.

getPicProc

A **Universal Procedure Pointer** to the low-level routine for retrieving information from the definition of a picture. The standard QuickDraw routine is the StdGetPic procedure.

putPicProc

A **Universal Procedure Pointer** to the low-level routine for saving information as the definition of a picture. The standard QuickDraw routine is the StdPutPic procedure.

Programming Info

C interface file: Quickdraw.h

QElem

A queue element for a Windows QHdr (IV–2345) structure.

```
struct QElem {  
    struct QElem *   qLink;  
    short             qType;  
    short             qData[1];  
};
```

qLink

A pointer to the next QElem structure.

qType

Undocumented

qData

Undocumented

Associated Functions

Dequeue (I–253)

Programming Info

C interface file: OSUtils.h

See Also

See QHdr (IV–2345).

QHdr

A Windows queue header structure.

```
struct QHdr {
    short    qFlags;
    short    pad;
    long     MutexID;
    QElemPtr qHead;
    QElemPtr qTail;
};
```

qFlags

Undocumented

pad

Unused.

MutexID

Undocumented

qHead

Undocumented

qTail

Undocumented

Associated Functions

CDSequenceSetSourceDataQueue (I-87)

Programming Info

C interface file: OSUtils.h

See Also

See QElem (IV-2344).

QTAItComponentCheckRecord

Provides component information for an 'rmcd' (IV-2576) reference movie component check atom.

```
struct QTAItComponentCheckRecord {
    long            flags;
    ComponentDescription cd;
    unsigned long   minVersion;
};
```

QTAltCPURatingRecord

flags

Unused; set to 0.

cd

A ComponentDescription (IV-2212) data structure giving the attributes of a class of components.

minVersion

The minimum version number of the required component.

Programming Info

C interface file: MoviesFormat.h

See Also

See 'rmcd' (IV-2576).

QTAltCPURatingRecord

Provides the user's CPU speed information for an 'rmcs' (IV-2577) reference movie CPU rating atom.

```
struct QTAltCPURatingRecord {  
    UInt32    flags;  
    UInt16    speed;  
};
```

flags

Unused; set to 0.

speed

A CPU speed constant (see below).

speed Constants

kQTCPU Speed1Rating

Slowest CPU speed

kQTCPU Speed2Rating

Next-to-slowest CPU speed.

kQTCPU Speed3Rating

Medium CPU speed.

kQTCPU Speed4Rating

Next-to-fastest CPU speed.

kQTCPU Speed5Rating

Fastest CPU speed.

Programming Info

C interface file: `MoviesFormat.h`

See Also

See 'rmcs' (IV-2577).

QTAltDataRateRecord

Provides data rate information for an 'rmdr' (IV-2578) reference movie data rate atom.

```
struct QTAltDataRateRecord {
    long    flags;
    long    dataRate;
};
```

`flags`

Unused; set to 0.

`dataRate`

A data rate constant (see below).

dataRate Constants

`kDataRate144ModemRate`

A rate of 14.4 kbps.

`kDataRate288ModemRate`

A rate of 28.8 kbps.

`kDataRateISDNRate`

A rate of 56 kbps.

`kDataRateDualISDNRate`

A rate of 112 kbps.

`kDataRateT1Rate`

A rate of 1.5 Mbps.

`kDataRateInfiniteRate`

A rate higher than any given rate.

Programming Info

C interface file: `MoviesFormat.h`

See Also

See 'rmdr' (IV-2578).

QTAltLanguageRecord

QTAltLanguageRecord

Provides language information for an 'rmla' (IV–2579) reference movie language atom.

```
struct QTAltLanguageRecord {  
    long    flags;  
    short   language;  
};
```

flags

Unused; set to 0.

language

The movie's language. See “Localization Codes” (IV–2673).

Programming Info

C interface file: `MoviesFormat.h`

See Also

See 'rmla' (IV–2579).

QTAltVersionCheckRecord

Provides version information for an 'rmvc' (IV–2581) reference movie version check atom.

```
struct QTAltVersionCheckRecord {  
    long    flags;  
    OSType  gestaltTag;  
    UInt32  val1;  
    UInt32  val2;  
    short   checkType;  
};
```

flags

Unused; set to 0.

gestaltTag

A system capability code.

val1

A version number (see discussion below).

val2

A version number (see discussion below).

checkType

A checking mode constant (see below).

checkType Constants

kVersionCheckMin

Field va11 contains the minimum version number required.

kVersionCheckMask

When field va12 is used to mask the actual version number, the result is equal to field va11.

Discussion

By selecting the checkType constant and putting appropriate values into va11 and va12, you can check either for product version numbers later than a given release or for specific information within the actual version number.

Programming Info

C interface file: MoviesFormat.h

See Also

See 'rmvc' (IV-2581).

QTAtomSpec

Specifies an atom and its container.

```
struct QTAtomSpec {
    QTAtomContainer    container;
    QTAtom             atom;
};
```

container

A QT atom container.

atom

A QT atom.

Programming Info

C interface file: Movies.h

QTCallbackHeader

Specifies the application-defined function for a callback operation.

```
struct QTCallbackHeader {
```

QTCallbackHeader

```
    long    callBackFlags;  
    long    reserved1;  
    SInt8    qtPrivate[40];  
};
```

callBackFlags

Contains flags (see below) that your component can use to communicate scheduling information about the **callback event** to the Movie Toolbox. This scheduling information tells the Movie Toolbox what time base events your clock component needs to know about to support the callback event. All unused flags must be set to 0.

reserved1

Reserved. Do not use.

qtPrivate

Reserved. Do not use.

callBackFlags Constants

qtcBNeedsRateChanges

Indicates that your clock component needs to know about rate changes. If you set this flag to 1, the Movie Toolbox calls `CLockRateChanged` (I-97) whenever the rate of the **callback event**'s time base changes.

qtcBNeedsTimeChanges

Indicates that your clock component needs to know about time changes. If you set this flag to 1, the Movie Toolbox calls `CLockTimeChanged` (I-100) whenever a program changes the time value of the time base, or when the time value changes by an amount that is different from the time base's rate.

qtcBNeedsStartStopChanges

Indicates that your clock component needs to know about the time base's start and stop changes. If you set this flag to 1, the Movie Toolbox calls `CLockStartStopChanged` (I-99) whenever a program changes the start or stop time of the time base.

Discussion

Your clock component provides functions that allow programs to use this data structure, which specifies the callback function for an operation. Your application can obtain callback function identifiers by calling `CLockNewCallBack` (I-96).

Programming Info

C interface file: `Movies.h`

See Also

See `CLockNewCallBack` (I-96).

QTChapterInfoRecord

Stores information for an entry in a chapter list.

```
struct QTChapterInfoRecord {
    long        index;
    TimeValue    time;
    Str255      name;
};
```

index

An index number for the entry. The first chapter has index of 1.

time

The time in the movie that the entry marks. Set this field to -1 if no more chapter entries are available.

name

The readable name of the entry.

Programming Info

C interface file: `Movies.h`

QTCustomActionTargetRecord

Defines a target for `CallComponentExecuteWiredAction` (I-60).

```
struct QTCustomActionTargetRecord {
    Movie        movie;
    DoMCActionUPP doMCActionCallbackProc;
    long         callBackRefcon;
    Track        track;
    long         trackObjectRefCon;
    Track        defaultTrack;
    long         defaultObjectRefCon;
    long         reserved1;
    long         reserved2;
};
```

movie

A movie identifier obtained from such functions as `NewMovie` (II-1069), `NewMovieFromFile` (II-1080), and `NewMovieFromHandle` (II-1084).

doMCActionCallbackProc

A **Universal Procedure Pointer** that accesses a `DoMCActionProc` (III-2082) callback.

QTDoScriptRecord

callbackRefCon

A reference constant to be passed to the DoMCActionProc callback. Use this value to point to a data structure containing any information the callback needs.

track

A track identifier obtained from such functions as NewMovieTrack (II–1092) and GetMovieTrack (I–497).

trackObjectRefCon

Undocumented

defaultTrack

The identifier of a default track, obtained from such functions as NewMovieTrack (II–1092) and GetMovieTrack (I–497).

defaultObjectRefCon

Undocumented

reserved1

Reserved.

reserved2

Reserved.

Programming Info

C interface file: MediaHandlers.h

QTDoScriptRecord

Provides information to MCDoAction (II–801) when mcActionDoScript is passed to it.

```
struct QTDoScriptRecord {  
    long    scriptTypeFlags;  
    char *  command;  
    char *  arguments;  
};
```

scriptTypeFlags

Constants (see below) that define the type of script.

command

Pointer to the wired action command.

arguments

Pointer to the wired action arguments.

scriptTypeFlags Constants

- kScriptIsUnknownType
Value is 1L << 0.
- kScriptIsJavaScript
Value is 1L << 1,
- kScriptIsLingoEvent
Value is 1L << 2.
- kScriptIsVBEvent
Value is 1L << 3.
- kScriptIsProjectorCommand
Value is 1L << 4.

Version Notes

Introduced in QuickTime 4.1.

Programming Info

C interface file: Movies.h

See Also

See MCDoAction (II-801).

QTEvaluateExpressionRecord

Undocumented

```
struct QTEvaluateExpressionRecord {  
    QTAtomSpec    expressionSpec;  
    float *        expressionResult;  
};
```

expressionSpec

Undocumented

expressionResult

Undocumented

Programming Info

C interface file: Movies.h

QTEventRecord

Records a user event for QuickTime.

QTFetchParameterAsRecord

```
struct QTEventRecord {
    long    version;
    OSType  eventType;
    Point   where;
    long    flags;
};
```

version

Undocumented

eventType

Undocumented

where

The location of the cursor at the time the event was posted.

flags

Undocumented

Discussion

This structure is used by the kActionSendQTEventToSprite action.

Associated Functions

ActionsProc (III–2075)

Programming Info

C interface file: Movies.h

QTFetchParameterAsRecord

Undocumented

```
struct QTFetchParameterAsRecord {
    QTAtomSpec  paramListSpec;
    long        paramIndex;
    long        paramType;
    long        allowedFlags;
    void *      min;
    void *      max;
    void *      currentValue;
    void *      newValue;
    Boolean      isUnsignedValue;
};
```

paramListSpec

Undocumented

paramIndex
Undocumented

paramType
Undocumented

allowedFlags
Undocumented

min
Undocumented

max
Undocumented

currentValue
Undocumented

newValue
Undocumented

isUnsignedValue
Undocumented

Programming Info

C interface file: Movies.h

QTGetChapterTimeRecord

Undocumented

```
struct QTGetChapterTimeRecord {
    StringPtr    chapterName;
    TimeRecord   chapterTime;
};
```

chapterName
Undocumented

chapterTime
Undocumented

Programming Info

C interface file: Movies.h

QTGetCursorByIDRecord

QTGetCursorByIDRecord

Undocumented

```
struct QTGetCursorByIDRecord {
    short    cursorID;
    Handle    colorCursorData;
    long      reserved1;
};
```

cursorID

Undocumented

colorCursorData

Undocumented

reserved1

Undocumented

Programming Info

C interface file: Movies.h

QTGetExternalMovieRecord

Undocumented

```
struct QTGetExternalMovieRecord {
    long          targetType;
    StringPtr      movieName;
    long          movieID;
    MoviePtr       theMovie;
    MovieControllerPtr theController;
};
```

targetType

Set to kTargetMovieName or kTargetMovieID.

movieName

Undocumented

movieID

Undocumented

theMovie

Undocumented

theController

Undocumented

Programming Info

C interface file: Movies.h

QTMIDIPort

Provides information about a MIDI port.

```
struct QTMIDIPort {
    SynthesizerConnections    portConnections;
    Str63                     portName;
};
```

portConnections

A SynthesizerConnections (IV–2464) structure.

portName

The name of the output port.

Programming Info

C interface file: QuickTimeMusic.h

QTMIDIPortList

Contains an array of information structures about MIDI ports.

```
struct QTMIDIPortList {
    short    portCount;
    QTMIDIPort    port[1];
};
```

portCount

The number of structures in the port field.

port

An array of QTMIDIPort (IV–2357) structures.

Associated Functions

NAGetMIDIPorts (II–1025)

Programming Info

C interface file: QuickTimeMusic.h

QTMovieExportSourceInfo

See Also

See QTMIDIPort (IV–2357).

QTMovieExportSourceInfo

Defines a single movie export data source by type, minimum and maximum sources of that type, and flags for that entry.

```
struct QTMovieExportSourceInfo {
    OSType      mediaType;
    UInt16      minCount;
    UInt16      maxCount;
    long        flags;
};
```

mediaType

Media type of the source; see “Codec Identifiers” (IV–2655).

minCount

Minimum number of sources of this kind required; 0 if none are required.

maxCount

Maximum number of sources of this kind allowed; –1 if unlimited sources are allowed.

flags

Reserved for flags.

Programming Info

C interface file: QuickTimeComponents.h

QTMovieExportSourceRecord

Contains an array of QTMovieExportSourceInfo (IV–2358) structures.

```
struct QTMovieExportSourceRecord {
    long          count;
    long          reserved;
    QTMovieExportSourceInfo  sourceArray[1];
};
```

count

The number of structures in the sourceArray field.

reserved

Reserved; do not use.

sourceArray

An array of QTMovieExportSourceInfo (IV–2358) structures.

Discussion

A 'src#' **resource** may be associated with export components that implement MovieExportFromProceduresToDataRef (II–922). The resource is used to indicate the types of data sources the export component can support. For each type, it also indicates the minimum and maximum number of data sources of that type. Clients can use this information to determine if the number of data sources they want to export can be handled directly by the exporter. If the data source type is supported by fewer sources are allowed, the client application must either export a fewer number of sources or combine the data from its candidate sources itself to meet the limitation imposed by the exporter.

Programming Info

C interface file: QuickTimeComponents.h

See Also

See QTMovieExportSourceInfo (IV–2358).

QTParamDialogEventRecord

Contains information about a user event in a dialog box.

```
struct QTParamDialogEventRecord {
    EventRecord *   theEvent;
    DialogPtr       whichDialog;
    short           itemHit;
};
```

theEvent

Pointer to an EventRecord (IV–2246) structure that describes the event received by the dialog box.

whichDialog

Pointer to the dialog box that received the event.

itemHit

Number of the dialog item that was hit.

Programming Info

C interface file: Movies.h

QTParmFetchPreviewRecord

QTParmFetchPreviewRecord

Contains information that defines a **preview**.

```
struct QTParmFetchPreviewRecord {  
    GWorldPtr    theWorld;  
    Fixed        percentage;  
};
```

theWorld

A pointer to the CGrafPort (IV–2168) that defines the graphics world into which to draw the **preview**.

percentage

The frame portion to be drawn, from 0.0 to 1.0.

Programming Info

C interface file: Movies.h

QTParmPreviewRecord

Contains information about a **preview** image.

```
struct QTParmPreviewRecord {  
    long          sourceID;  
    PicHandle     sourcePicture;  
};
```

sourceID

The number of the **preview** image.

sourcePicture

The **preview** image.

Programming Info

C interface file: Movies.h

QTPresetInfo

Provides data for a QTPresetListRecord (IV–2361) structure.

```
struct QTPresetInfo {  
    OSType        presetKey;  
    UInt32        presetFlags;
```

```

    OSType      settingsResourceType;
    SInt16      settingsResourceID;
    SInt16      padding1;
    SInt16      nameStringListID;
    SInt16      nameStringIndex;
    SInt16      infoStringListID;
    SInt16      infoStringIndex;
};

```

presetKey

The unique key for this structure in the QTPresetListRecord (IV–2361) structure.

presetFlags

Flags about this preset

settingsResourceType

Resource type of settings **resource**.

settingsResourceID

Resource id of settings **resource**

padding1

Unused.

nameStringListID

Name string list **resource** ID.

nameStringIndex

Name string index.

infoStringListID

Info string list **resource** ID

infoStringIndex

Info string index.

Programming Info

C interface file: QuickTimeComponents.h

QTPresetListRecord

Contains an array of QTPresetInfo (IV–2360) structures.

```

struct QTPresetListRecord {
    UInt32      flags;
    UInt32      count;
    UInt32      reserved;
    QTPresetInfo presetsArray[1];
};

```

QTResolutionSettings

```
};

flags
    Flags for the whole list.
count
    The number of elements in the presetsArray field.
reserved
    Reserved.
presetsArray
    An array of QTPresetInfo (IV-2360) structures.
```

Programming Info

C interface file: QuickTimeComponents.h

QTResolutionSettings

Provides data for a 'reso' (IV-2576) atom.

```
struct QTResolutionSettings {
    Fixed    horizontalResolution;
    Fixed    verticalResolution;
};

horizontalResolution
    The horizontal resolution, in dots per inch.
verticalResolution
    The vertical resolution, in dots per inch.
```

Programming Info

C interface file: ImageCompression.h

QTRestartAtTimeRecord

Undocumented

```
struct QTRestartAtTimeRecord {
    TimeValue    startTime;
    Fixed        rate;
};
```


startTime
The start time in the movie time scale.

rate
The playback rate. If the rate is 0, the movie’s current rate is maintained.

Programming Info
C interface file: ImageCompression.h

QTRuntimeSpriteDescStruct

Provides a **sprite** description for the SpriteMediaNewSprite (III–1901) function.

```
QTRuntimeSpriteDescStruct {  
    long                version;  
    QTAtomID            spriteID;  
    short               imageIndex;  
    MatrixRecord        matrix;  
    short               visible;  
    short               layer;  
    ModifierTrackGraphicsModeRecord graphicsMode;  
    QTAtomID            actionHandlingSpriteID;  
};
```

version
Set to 0.

spriteID
The QT atom ID of the **sprite** atom.

imageIndex
The index of the **sprite** image. This value must be between 1 and the number of available images. You can determine how many images are available by calling SpriteMediaCountImages (III–1886).

matrix
A MatrixRecord (IV–2304) structure that defines the **sprite**’s matrix.

visible
Undocumented

layer
The **sprite**’s layer number.

graphicsMode
A ModifierTrackGraphicsModeRecord (IV–2311) structure that defines the graphics mode setting for the **sprite**.

QTSAtomContainerDataStruct

actionHandlingSpriteID

Undocumented

Programming Info

C interface file: Movies.h

QTSAtomContainerDataStruct

Undocumented

```
struct QTSAtomContainerDataStruct {
    QTAtomContainer    container;
    QTAtom             parentAtom;
};
```

container

A QT atom container.

parentAtom

The container's parent atom.

Programming Info

C interface file: QuickTimeStreaming.h

QTSAudioParams

Provides audio data for the QTSMediaParams (IV-2374) structure.

```
struct QTSAudioParams {
    SInt16    leftVolume;
    SInt16    rightVolume;
    SInt16    bassLevel;
    SInt16    trebleLevel;
    short     frequencyBandsCount;
    void *     frequencyBands;
    Boolean    levelMeteringEnabled;
};
```

leftVolume

The playback volume for the left channel, where 0x0000 represents no volume and 0x0100 represents full volume.

rightVolume

The playback volume for the right channel, where 0x0000 represents no volume and 0x0100 represents full volume.

bassLevel

Undocumented

trebleLevel

Undocumented

frequencyBandsCount

Undocumented

frequencyBands

Undocumented

levelMeteringEnabled

TRUE if level metering is enabled, FALSE otherwise.

Programming Info

C interface file: QuickTimeStreaming.h

QTSBandwidthAlertParams

Undocumented

```
struct QTSBandwidthAlertParams {
    long          flags;
    TimeValue     restartAt;
    void *        reserved;
};
```

flags

Undocumented

restartAt

Undocumented

reserved

Undocumented

Programming Info

C interface file: QuickTimeStreaming.h

QTSTBufferTimeAtom

QTSTBufferTimeAtom

Undocumented

```
struct QTSTBufferTimeAtom {  
    SInt32    versionAndFlags;  
    Fixed     bufferTime;  
};
```

versionAndFlags

Undocumented

bufferTime

Undocumented

Programming Info

C interface file: QuickTimeStreaming.h

QTSTCanHandleSendDataTypeParams

Provides data type capability information to the QTSPresGetInfo (II–1272) structure.

```
struct QTSTCanHandleSendDataTypeParams {  
    SInt32    modifierTypeOrInputID;  
    Boolean    isModifierType;  
    Boolean    returnedCanHandleSendDataType;  
};
```

modifierTypeOrInputID

A modifier type or input ID.

isModifierType

True if modifierTypeOrInputID is a modifier type, FALSE if it is an input ID.

returnedCanHandleSendDataType

The called component sets this field to TRUE if it can handle the data type.

Programming Info

C interface file: QuickTimeStreaming.h

QTScheduledBandwidthRecord

Provides information to the QTScheduledBandwidthRequest (II–1219) function.

```
struct QTScheduledBandwidthRecord {
    long      recordSize;
    long      priority;
    long      dataRate;
    CompTimeValue startTime;
    CompTimeValue duration;
    TimeScale  scale;
    TimeBase   base;
};
```

recordSize
The number of bytes in this structure.

priority
Undocumented

dataRate
The data rate.

startTime
The **band**width usage start time.

duration
Duration of **band**width usage, or 0 if unknown.

scale
The timescale of the duration field.

base
The time base.

Programming Info

C interface file: Movies.h

QTSClipRectAtom

Undocumented

```
struct QTSClipRectAtom {
    Sint32  versionAndFlags;
    Rect     clipRect;
};
```

versionAndFlags
Undocumented

clipRect
A Rect (IV–2399) structure that defines a clipping rectangle.

QTSDDataRateHistoryParams

Programming Info

C interface file: QuickTimeStreaming.h

QTSDDataRateHistoryParams

Provides data rate history information to the QTSPresGetInfo (II–1272) function.

```
struct QTSDDataRateHistoryParams {
    UInt32      changeSeed;
    UInt32      millisecondsBetweenElements;
    UInt32      numberOfIntervals;
    UInt32 *    elements;
};
```

changeSeed

This value changes if the information in the structure changes.

millisecondsBetweenElements

Undocumented

numberOfIntervals

Undocumented

elements

Pointer to *Undocumented* in bits per second; callee allocates; caller must dispose.

Programming Info

C interface file: QuickTimeStreaming.h

QTSDDimensionParams

Provides source dimension information to the QTSPresGetInfo (II–1272) function.

```
struct QTSDimensionParams {
    Fixed      width;
    Fixed      height;
};
```

width

The width in pixels.

height

The height in pixels.

Programming Info

C interface file: QuickTimeStreaming.h

QTSDirectConnectPrefsRecord

Undocumented

```
struct QTSDirectConnectPrefsRecord {
    UInt32    tcpPortID;
    OSType    protocol;
};
```

tcpPortID

Undocumented

protocol

*Undocumented***Programming Info**

C interface file: QuickTimeStreaming.h

QTSDurationAtom

Provides duration information to the QTSPresGetInfo (II-1272) function.

```
struct QTSDurationAtom {
    SInt32    versionAndFlags;
    TimeScale    timeScale;
    TimeValue64    duration;
};
```

versionAndFlags

Version information and flags.

timeScale

The time scale.

duration

The duration.

Programming Info

C interface file: QuickTimeStreaming.h

QTSEditEntry

QTSEditEntry

Provides data to the QTSEditList (IV–2370) structure.

```
struct QTSEditEntry {  
    TimeValue64    presentationDuration;  
    TimeValue64    streamStartTime;  
    Fixed          streamRate;  
};
```

presentationDuration

The duration of the presentation.

streamStartTime

Stream starting time.

streamRate

Streaming rate.

Programming Info

C interface file: QuickTimeStreaming.h

QTSEditList

Contains an array of QTSEditEntry (IV–2370) structures.

```
struct QTSEditList {  
    SInt32          numEdits;  
    QTSEditEntry    edits[1];  
};
```

numEdits

The number of structures in the edits field.

edits

An array of QTSEditEntry (IV–2370) structures.

Programming Info

C interface file: QuickTimeStreaming.h

See Also

See QTSEditEntry (IV–2370).

QTSErrorParams

Undocumented

```
struct QTSErrorParams {
    const char *   errorString;
    Sint32         flags;
};
```

errorString

A pointer to an error message.

flags

Undocumented

Programming Info

C interface file: QuickTimeStreaming.h

QTSettingsVersionAtomRecord

Format of the 'vers' atom found in settings atom containers.

```
struct QTSettingsVersionAtomRecord {
    long    componentVersion;
    short   flags;
    short   reserved;
};
```

componentVersion

Standard compression component version.

flags

The low bit is 1 if the platform is **little-endian**, 0 if it is **big-endian**.

reserved

Reserved; set to 0.

Programming Info

C interface file: QuickTimeComponents.h

QTSGetURLLinkRecord

Holds URL information for QTSPresGetInfo (II-1272) and QTSPresSetInfo (II-1293).

QTSGraphicsModeParams

```
struct QTSGetURLLinkRecord {
    Point      displayWhere;
    Handle     returnedURLLink;
};
```

displayWhere
Undocumented

returnedURLLink
Undocumented

Programming Info

C interface file: QuickTimeStreaming.h

QTSGraphicsModeParams

Defines a source's graphics mode.

```
struct QTSGraphicsModeParams {
    SInt16      graphicsMode;
    RGBColor     opColor;
};
```

graphicsMode

The graphics mode; see “Graphics Transfer Modes” (IV–2670).

opColor

The color for use in addPin, subPin, blend, and transparent operations, as an RGBColor (IV–2403) structure. If NIL, the source's opcolor is left unchanged.

Programming Info

C interface file: QuickTimeStreaming.h

QTSInfoParams

Undocumented

```
struct QTSInfoParams {
    OSType      infoType;
    void *      infoParams;
};
```

infoType

Undocumented

infoParams

*Undocumented***Programming Info**

C interface file: QuickTimeStreaming.h

QTSLostPercentParams

Records a stream's lost data percentage, combined with the existing percentage, for QTSPresGetInfo (II-1272) and QTSPresSetInfo (II-1293).

```
struct QTSLostPercentParams {
    UInt32    receivedPkts;
    UInt32    lostPkts;
    Fixed     percent;
};
```

receivedPkts

The number of packets received.

lostPkts

The number of packets lost.

percent

The percent of packets lost.

Programming Info

C interface file: QuickTimeStreaming.h

QTSMediaIndSampleDescriptionParams

Identifies sample data for QTSMediaGetInfo (II-1245), QTSMediaSetInfo (II-1248), QTSMediaGetIndStreamInfo (II-1244), and QTSMediaSetIndStreamInfo (II-1246).

```
struct QTSMediaIndSampleDescriptionParams {
    SInt32          index;
    OSType          returnedMediaType;
    SampleDescriptionHandle returnedSampleDescription;
};
```

index

An index value indicating which of the media's data references contains the sample data for this sample description.

QTSMediaNotificationParams

returnedMediaType

A media type; see “Data References” (IV–2661).

returnedSampleDescription

A handle to a SampleDescription (IV–2414) structure.

Programming Info

C interface file: QTSMovie.h

QTSMediaNotificationParams

Identifies the notification process for QTSMediaGetInfo (II–1245), QTSMediaSetInfo (II–1248), QTSMediaGetIndStreamInfo (II–1244), and QTSMediaSetIndStreamInfo (II–1246).

```
struct QTSMediaNotificationParams {  
    QTSNotificationUPP    notificationProc;  
    void *                notificationRefCon;  
    SInt32                flags;  
};
```

notificationProc

A **Universal Procedure Pointer** that accesses a QTSNotificationProc (III–2126) callback.

notificationRefCon

A pointer to data that will be passed to the callback.

flags

Undocumented

Programming Info

C interface file: QTSMovie.h

QTSMediaParams

Combines the QTSVideoParams (IV–2389) and QTSAudioParams (IV–2364) structures.

```
struct QTSMediaParams {  
    QTSVideoParams    v;  
    QTSAudioParams    a;  
};
```

v
A QTSTVideoParams (IV–2389) structure.

a
A QTSAudioParams (IV–2364) structure.

Programming Info

C interface file: QuickTimeStreaming.h

QTSMediaPresentationParams

Identifies a presentation for QTSMediaGetInfo (II–1245), QTSMediaSetInfo (II–1248), QTSMediaGetIndStreamInfo (II–1244), and QTSMediaSetIndStreamInfo (II–1246).

```
struct QTSMediaPresentationParams {
    QTSPresentation    presentationID;
};
```

presentationID

A pointer to a QTSPresentationRecord (IV–2379) structure.

Programming Info

C interface file: QTSMovie.h

QTSMediaStreamHeaderAtom

Undocumented

```
struct QTSMediaStreamHeaderAtom {
    Sint32    versionAndFlags;
    OSType    mediaTransportType;
    OSType    mediaTransportDataAID;
};
```

versionAndFlags

Undocumented

mediaTransportType

Undocumented

mediaTransportDataAID

The data for this structure.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNameParams

QTSNameParams

Holds a presentation or stream name for QTSPresGetInfo (II–1272).

```
struct QTSNameParams {  
    SInt32          maxNameLength;  
    SInt32          requestedLanguage;  
    SInt32          returnedActualLanguage;  
    unsigned char * returnedName;  
};
```

maxNameLength

The maximum number of bytes to render the name.

requestedLanguage

The requested language for the name; see “Localization Codes” (IV–2673).

returnedActualLanguage

On return, the actual language used for the name; see “Localization Codes” (IV–2673).

returnedName

Pointer to a Pascal string supplied by the caller.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNewPresDetectedParams

Undocumented

```
struct QTSNewPresDetectedParams {  
    void * data;  
};
```

data

Pointer to data.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNewPresentationParams

Specifies a presentation for QTSNewPresentation (II–1250).

```

struct QTSNewPresentationParams {
    OSType          dataType;
    const void *    data;
    UInt32          dataLength;
    QTSEditListHandle editList;
    SInt32          flags;
    TimeScale       timeScale;
    QTSMediaParams * mediaParams;
    QTSNotificationUPP notificationProc;
    void *          notificationRefCon;
};

```

dataType

Undocumented

data

Undocumented

dataLength

Undocumented

editList

A handle to a QTSEditList (IV–2370) structure.

flags

Undocumented

timeScale

The time scale; set to 0 for the default time scale.

mediaParams

Undocumented

notificationProc

A pointer to a QTSNotificationProc (III–2126) callback.

notificationRefCon

A reference constant to be passed to the QTSNotificationProc callback.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNewStreamParams

Undocumented

```

struct QTSNewStreamParams {
    QTSSStream  stream;
};

```

QTSNoProxyPref

```
};
```

```
stream
```

A pointer to a QTSSStreamRecord (IV–2387) structure.

Programming Info

C interface file: QuickTimeStreaming.h

QTSNoProxyPref

Provides data for the QTSPrefsGetNoProxyURLs (II–1263) function.

```
struct QTSNoProxyPref {  
    UInt32    flags;  
    UInt32    seed;  
    char      urlList[1];  
};
```

```
flags
```

Undocumented

```
seed
```

A seed value from the last time this setting was read from the system preferences.

```
urlList
```

A null-terminated, comma-delimited list of URLs.

Associated Functions

QTSPrefsGetNoProxyURLs (II–1263)

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresentationHeaderAtom

Undocumented

```
struct QTSPresentationHeaderAtom {  
    SInt32    versionAndFlags;  
    OSType    conductorOrDataType;  
    OSType    dataAtomType;  
};
```


versionAndFlags

Undocumented

conductorOrDataType

Undocumented

dataAtomType

Where the data is actually located.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresentationRecord

Defines a presentation.

```
struct QTSPresentationRecord {
    long    data[1];
};
```

data

Array of data that constitutes the presentation.

Programming Info

C interface file: QuickTimeStreaming.h

QTSPresIdleParams

Provides parameters for QTSPresIdle (II-1284).

```
struct QTSPresIdleParams {
    QTSSStream    stream;
    TimeValue64   movieTimeToDisplay;
    SInt32         flagsIn;
    SInt32         flagsOut;
};
```

stream

A pointer to a QTSSStreamRecord (IV-2387) structure.

movieTimeToDisplay

Undocumented

flagsIn

Undocumented

QTSpriteButtonBehaviorStruct

flagsOut

Undocumented

Associated Functions

QTSPresIdle (II–1284)

Programming Info

C interface file: QuickTimeStreaming.h

See Also

See QTSPresIdle (II–1284).

QTSpriteButtonBehaviorStruct

Contains ID information for four **sprite** state atoms.

```
struct QTSpriteButtonBehaviorStruct {
    QTAtomID    notOverNotPressedStateID;
    QTAtomID    overNotPressedStateID;
    QTAtomID    overPressedStateID;
    QTAtomID    notOverPressedStateID;
};
```

notOverNotPressedStateID

Undocumented

overNotPressedStateID

Undocumented

overPressedStateID

Undocumented

notOverPressedStateID

Undocumented

Programming Info

C interface file: Movies.h

QTSProxyPref

Provides data for the QTSPrefsFindProxyByType (II–1260) function.

```
struct QTSProxyPref {
    UInt32    flags;
    SInt32    portID;
    UInt32    seed;
```

```
    Str255    serverNameStr;
};
```

flags

Undocumented

portID

ID of the port to use for this connection type.

seed

A seed value from the last time this setting was read from the system preferences.

serverNameStr

A proxy server URL.

Associated Functions

QTSPrefsFindProxyByType (II-1260)

Programming Info

C interface file: QuickTimeStreaming.h

QTSPProxyPrefsRecord

Undocumented

```
struct QTSPProxyPrefsRecord {
    Str255    serverNameStr;
    UInt32    portID;
};
```

serverNameStr

A proxy server URL.

portID

ID of the port to use for this connection type.

Programming Info

C interface file: QuickTimeStreaming.h

QTSSampleDescription

Undocumented

```
struct QTSSampleDescription {
```

QTStatHelperNextParams

```
long      descSize;  
long      dataFormat;  
long      resvd1;  
short     resvd2;  
short     dataRefIndex;  
UInt32    version;  
UInt32    resvd3;  
SInt32    flags;
```

descSize

Undocumented

dataFormat

Undocumented

resvd1

Reserved; set to 0.

resvd2

Reserved; set to 0.

dataRefIndex

Undocumented

version

Undocumented

resvd3

Reserved set to 0.

flags

Undocumented

Programming Info

C interface file: QTSMovie.h

QTStatHelperNextParams

Holds information about the next streaming statistic obtained by QTStatHelperNext (II-1311).

```
struct QTStatHelperNextParams {  
    SInt32      flags;  
    OSType      returnedStatisticsType;  
    QTSSStream  returnedStream;  
    UInt32      maxStatNameLength;  
    char *      returnedStatName;  
    UInt32      maxStatStringLength;
```

```

    char *      returnedStatString;
    UInt32      maxStatUnitLength;
    char *      returnedStatUnit;
};

```

flags

Undocumented

returnedStatisticsType

Undocumented

returnedStream

On return, a pointer to a QTStreamRecord (IV-2387) structure.

maxStatNameLength

Undocumented

returnedStatName

Undocumented; pass NIL if you don't want this information.

maxStatStringLength

Undocumented

returnedStatString

Undocumented; pass NIL if you don't want this information.

maxStatUnitLength

Undocumented

returnedStatUnit

Undocumented; pass NIL if you don't want this information.

flags Constants

kQTStatHelperReturnPascalStringsFlag

Undocumented

Discussion

When you call QTStatHelperNext (II-1311), specifying a statistic helper and the address of this structure, QuickTime fills in this structure with information about the next statistic obtained by the statistic helper.

Associated Functions

QTStatHelperNext (II-1311)

Programming Info

C interface file: QuickTimeStreaming.h

QTStatHelperRecord

QTStatHelperRecord

Defines the component instance of a statistics helper.

```
struct QTStatHelperRecord {  
    long    data[1];  
};
```

data

The component instance of the statistics helper.

Programming Info

C interface file: QuickTimeStreaming.h

QTStatisticsParams

Defines stream statistics for the QTSPresGetInfo (II-1272) function.

```
struct QTStatisticsParams {  
    OSType          statisticsType;  
    QTAtomContainer container;  
    QTAtom          parentAtom;  
    SInt32          flags;  
};
```

statisticsType

A constant (see below) that defines the type of statistics.

container

A QT atom container.

parentAtom

The parent atom of the container parameter.

flags

Undocumented

statisticsType Constants

kQTSA11StatisticsType

A full statistics helper for all statistics; constant value is 'all'.

kQTSShortStatisticsType

A short statistics helper; constant value is 'shrt'.

kQTSSummaryStatisticsType

A summary statistics helper; constant value is 'summ'.

Programming Info

C interface file: QuickTimeStreaming.h

QTSSStatusParams

Undocumented

```

struct QTSSStatusParams {
    UInt32      status;
    const char * statusString;
    UInt32      detailedStatus;
    const char * detailedStatusString;
};

```

status

Undocumented

statusString

Undocumented

detailedStatus

Undocumented

detailedStatusString

*Undocumented***Programming Info**

C interface file: QuickTimeStreaming.h

QTSSStreamBuffer

Defines a stream buffer for QuickTime streaming.

```

struct QTSSStreamBuffer {
    struct QTSSStreamBuffer * reserved1;
    struct QTSSStreamBuffer * reserved2;
    struct QTSSStreamBuffer * next;
    unsigned char *          rptr;
    unsigned char *          wptr;
    long                    reserved3;
    UInt32                  metadata[4];
    SInt32                   flags;
};

```

QTSSStreamChangedParams

reserved1

Reserved; do not use.

reserved2

Reserved; do not use.

next

A pointer to the next message block in a message.

rpPtr

A pointer to the first byte in the data buffer that contains real data.

wPtr

A pointer to the byte after the last byte in the data buffer that contains real data.

reserved3

Reserved; do not use.

metadata

Usage defined by message sender.

flags

Reserved; do not use.

Associated Functions

QTSCopyMessage (II–1220)

Programming Info

C interface file: QuickTimeStreaming.h

QTSSStreamChangedParams

Undocumented

```
struct QTSSStreamChangedParams {  
    QTSSStream      stream;  
    ComponentInstance mediaComponent;  
};
```

stream

A pointer to a QTSSStreamRecord (IV–2387) structure.

mediaComponent

Undocumented; could be NIL.

Programming Info

C interface file: QuickTimeStreaming.h

QTSSStreamGoneParams

Undocumented

```
struct QTSSStreamGoneParams {
    QTSSStream    stream;
};
```

stream

A pointer to a QTSSStreamRecord (IV–2387) structure.

Programming Info

C interface file: QuickTimeStreaming.h

QTSSStreamRecord

Contains a stream for QuickTime streaming.

```
struct QTSSStreamRecord {
    long    data[1];
};
```

data

An array of data representing the stream.

Programming Info

C interface file: QuickTimeStreaming.h

QTStatusStringRecord

Passes status information to an MCActionFilterProc (III–2096) callback.

```
struct QTStatusStringRecord {
    long    stringTypeFlags;
    char *   statusString;
};
```

stringTypeFlags

Flags (see below) that define the string type.

statusString

A pointer to the status string.

QTSTransportPref

stringTypeFlags Constants

- kStatusStringIsURLLink
The string is a URL.
- kStatusStringIsStreamingStatus
Undocumented
- kStatusHasCodeNumber
The high 16 bits is an error code number.
- kStatusIsError
Undocumented

Programming Info

C interface file: `Movies.h`

QTSTransportPref

Records streaming transport preferences.

```
struct QTSTransportPref {  
    OSType    protocol;  
    SInt32    portID;  
    UInt32    flags;  
    UInt32    seed;  
};
```

`protocol`

Constant that identifies the streaming transport protocol; see “Streaming Transport Atoms” (IV–2695).

`portID`

ID of the port to use for this connection type.

`flags`

Connection flags (see below).

`seed`

A seed value from the last time this setting was read from the system preferences.

flags Constants

- kConnectionActive
The connection is active.
- kConnectionUseSystemPref
Use the system preferences.

Associated Functions

QTSPrefsFindConnectionByType (II–1259)

Programming Info

C interface file: QuickTimeStreaming.h

QTSURLParams

Undocumented

```
struct QTSURLParams {
    UInt32      urlLength;
    const char * url;
};
```

urlLength

The length of the URL.

url

Pointer to a URL.

Programming Info

C interface file: QuickTimeStreaming.h

QTSVideoParams

Provides video parameter data for the QTSMediaParams (IV–2374) structure.

```
struct QTSVideoParams {
    Fixed      width;
    Fixed      height;
    MatrixRecord matrix;
    CGrafPtr   gWorld;
    GDHandle    gdHandle;
    RgnHandle   clip;
    short      graphicsMode;
    RGBColor    opColor;
};
```

width

The image width in pixels.

height

The image height in pixels.

QTSVolumesParams

`matrix`

A `MatrixRecord` (IV–2304) structure representing the video matrix.

`gWorld`

A pointer to the `CGrafPort` (IV–2168) structure for the video.

`gdHandle`

A handle to the `GDevice` (IV–2264) structure for the video.

`clip`

A handle to `MacRegion` (IV–2303) structure that defines the video clipping region.

`graphicsMode`

The video’s graphics mode; see “Graphics Transfer Modes” (IV–2670).

`opColor`

An `RGBColor` (IV–2403) structure that defines the color for use in `addPin`, `subPin`, `blend`, and transparent operations. If `NIL`, the source’s `opcolor` is left unchanged.

Programming Info

C interface file: `QuickTimeStreaming.h`

QTSVolumesParams

Stores source sound volumes for streaming.

```
struct QTSVolumesParams {
    SInt16    leftVolume;
    SInt16    rightVolume;
};
```

`leftVolume`

The playback volume for the left channel, where `0x0000` represents no volume and `0x0100` represents full volume.

`rightVolume`

The playback volume for the right channel, where `0x0000` represents no volume and `0x0100` represents full volume.

Programming Info

C interface file: `QuickTimeStreaming.h`

QTSyncTaskRecord

Undocumented

```
struct QTSyncTaskRecord {
    void *      qLink;
    QTSyncTaskUPP  proc;
};
```

qLink

Undocumented

proc

A **Universal Procedure Pointer** that accesses a QTSyncTaskProc (III–2127) callback.

Programming Info

C interface file: Movies.h

QTTargetDataSize

Contains a setting for a 'dasz' (IV–2527) atom.

```
struct QTTargetDataSize {
    unsigned long  targetDataSize;
};
```

targetDataSize

The target data size setting.

Discussion

The JPEG graphics exporter supports the target data size setting, which is implemented by compressing the image repeatedly, lowering the quality until the target size (or a maximum attempt limit) is reached.

Programming Info

C interface file: ImageCompression.h

See Also

See the 'dasz' (IV–2527) atom.

QTTweenerRecord

Stores a **tween** for the QTNewTween (II–1209) function.

QTVRAreaOfInterest

```
struct QTweenerRecord {  
    long    data[1];  
};
```

data

An array of data that constitutes a **tween**.

Programming Info

C interface file: Movies.h

QTVRAreaOfInterest

Contains information passed to QTVRSetBackBufferImagingProc (II-1411).

```
struct QTVRAreaOfInterest {  
    float    panAngle;  
    float    tiltAngle;  
    float    width;  
    float    height;  
    UInt32   flags;  
};
```

panAngle

The pan angle of the upper-left coordinate (in panorama space) of the area of interest.

tiltAngle

The tilt angle of the upper-left coordinate (in panorama space) of the area of interest.

width

The width of the area of interest.

height

The height of the area of interest.

flags

A set of bit flags (see below) that indicate when to call the back buffer imaging procedure for this area of interest.

flags Constants

kQTVRBackBufferEveryUpdate

If this bit is set, the back buffer imaging procedure is to be called whenever QuickTime is about to update the window containing the specified QuickTime VR movie instance. That is, the procedure is called just before QuickTime

unwarps the back buffer image into the prescreen buffer and redraws the screen image.

kQTVRBackBufferEveryIdle

If this bit is set, the back buffer imaging procedure is to be called when either `MCIsPlayerEvent` (II–819) is called with the `idle` parameter, or when `MCIdle` (II–816) is called. Its purpose is to cause the software to draw as often as possible.

kQTVRBackBufferAlwaysRefresh

If this bit is set, the back buffer is always refreshed to the proper movie data just before your back buffer imaging procedure is called. If your back buffer imaging procedure completely overwrites the rectangle passed to it, you should not set this bit.

Discussion

The `areasOfInterest` parameter to `QTVRSetBackBufferImagingProc` (II–1411) specifies an array of `QTVRAreaOfInterest` structures, each one of which indicates a rectangular area in the QTVR back buffer.

Associated Functions

`QTVRSetBackBufferImagingProc` (II–1411)

Programming Info

C interface file: `QuickTimeVR.h`

See Also

See `QTVRSetBackBufferImagingProc` (II–1411).

QTVRCubicFaceData

Non-standard cubic panorama data for a 'cuFa' (IV–2525) atom.

```
struct QTVRCubicFaceData {
    float    orientation[4];
    float    center[2];
    float    aspect;
    float    skew;
};
```

`orientation`
Undocumented.

`center`
Undocumented.

QTVRCubicViewAtom

aspect

Undocumented.

skew

Undocumented.

Programming Info

C interface file: QuickTimeVR.h

See Also

See 'cuva' (IV–2525).

QTVRCubicViewAtom

Cubic view data for a 'cuvw' (IV–2526) atom.

```
struct QTVRCubicViewAtom {  
    Float32    minPan;  
    Float32    maxPan;  
    Float32    minTilt;  
    Float32    maxTilt;  
    Float32    minFieldOfView;  
    Float32    maxFieldOfView;  
    Float32    defaultPan;  
    Float32    defaultTilt;  
    Float32    defaultFieldOfView;  
};
```

minPan

Undocumented

maxPan

Undocumented

minTilt

Undocumented

maxTilt

Undocumented

minFieldOfView

Undocumented

maxFieldOfView

Undocumented

defaultPan

Undocumented

defaultTilt

Undocumented

defaultFieldOfView

*Undocumented***Programming Info**

C interface file: QuickTimeVR.h

See Also

See 'cuvw' (IV–2526).

STR

Data Structures

QTVRCursorRecord

Contains information passed to QTVRReplaceCursor (II–1407).

```

struct QTVRCursorRecord {
    UInt16    theType;
    SInt16    rsrcID;
    Handle    handle;
};

```

theType

A constant (see below) that defines the type of cursor to replace.

rsrcID

The **resource** ID of the cursor to replace; see “QTVR Cursors” (IV–2688).

handle

A handle to the cursor data that is to replace the specified cursor. If theType is kQTVRUseDefaultCursor, then this field should contain NIL.

theType Constants

kQTVRUseDefaultCursor

Restore the default cursor. In this case, the handle field of the cursor record should contain NIL.

kQTVRStdCursorType

The cursor is a standard black-and-white cursor.

kQTVRColorCursorType

The cursor is a color cursor.

Discussion

The cursRecord parameter to QTVRReplaceCursor (II–1407) specifies a QTVRCursorRecord structure, which indicates the cursor to replace and its replacement cursor.

QTVRFloatPoint

Version Notes

In earlier versions of QuickTime, theType was called the type field.

Associated Functions

QTVRReplaceCursor (II–1407)

Programming Info

C interface file: QuickTimeVR.h

See Also

See QTVRReplaceCursor (II–1407).

QTVRFloatPoint

Specifies a point in a QTVR panorama or object.

```
struct QTVRFloatPoint {  
    float    x;  
    float    y;  
};
```

x

The horizontal coordinate of the point.

y

The vertical coordinate of the point.

Associated Functions

QTVRAnglesToCoord (II–1334)

Programming Info

C interface file: QuickTimeVR.h

QTVRInterceptRecord

Contains information about a QTVRInterceptProc (III–2130) being intercepted by QTVR.

```
struct QTVRInterceptRecord {  
    SInt32    reserved1;  
    SInt32    selector;  
    SInt32    reserved2;  
    SInt32    reserved3;  
    SInt32    paramCount;  
    void *    parameter[6];  
};
```

reserved1

Reserved; do not use.

selector

A constant that indicates which routine has triggered your intercept procedure (see below).

reserved2

Reserved; do not use.

reserved3

Reserved; do not use.

paramCount

The number of items in the parameter field.

parameter

An array that holds, in order, the parameters that were passed to the intercepted function, minus the QTVR instance parameter. For example, if you intercept the QTVRSetPanAngle function, the parameter field contains a single item, a pointer to a floating-point value that is the new pan angle. You can determine how many items the parameter field contains by inspecting the paramCount field.

selector Constants

kQTVRSetPanAngleSelector

Value is 0x2000.

kQTVRSetTiltAngleSelector

Value is 0x2000.

kQTVRSetFieldOfViewSelector

Value is 0x2002.

kQTVRSetViewCenterSelector

Value is 0x2003.

kQTVRMouseEnterSelector

Value is 0x2004.

kQTVRMouseWithinSelector

Value is 0x2005.

kQTVRMouseLeaveSelector

Value is 0x2006.

kQTVRMouseDownSelector

Value is 0x2007.

QTVRPanoImagingAtom

kQTVRMouseDownSelector

Value is 0x2008.

kQTVRMouseUpSelector

Value is 0x2009.

kQTVRTriggerHotSpotSelector

Value is 0x200A.

kQTVRGetHotSpotTypeSelector

Value is 0x200B.

Associated Functions

QTVRCallInterceptedProc (II–1336)

Programming Info

C interface file: QuickTimeVR.h

See Also

See QTVRInterceptProc (III–2130).

QTVRPanoImagingAtom

Provides panorama imaging data for an 'impr' atom.

```
struct QTVRPanoImagingAtom {
    UInt16    majorVersion;
    UInt16    minorVersion;
    UInt32    imagingMode;
    UInt32    imagingValidFlags;
    UInt32    correction;
    UInt32    quality;
    UInt32    directDraw;
    UInt32    imagingProperties[6];
    UInt32    reserved1;
    UInt32    reserved2;
};
```

majorVersion

Undocumented.

majorVersion

Undocumented.

imagingMode

Undocumented.

imagingValidFlags

Undocumented.

correction

Undocumented.

quality

Undocumented.

directDraw

Undocumented.

imagingProperties

Reserved for future properties.

reserved1

Reserved; do not use.

reserved2

Reserved; do not use.

Programming Info

C interface file: QuickTimeVRFormat.h

See Also

See the 'impn' (IV-2551) atom.

Rect

Defines the size and location of a QuickDraw rectangle.

```
struct Rect {
    short    top;
    short    left;
    short    bottom;
    short    right;
};
```

top

The vertical coordinate of the upper-left point of the rectangle.

left

The horizontal coordinate of the upper-left point of the rectangle.

bottom

The vertical coordinate of the lower-right point of the rectangle.

ReferenceMovieDataRefRecord

right

The horizontal coordinate of the lower-right point of the rectangle.

Programming Info

C interface file: Quickdraw.h

See Also

See *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

ReferenceMovieDataRefRecord

Provides a reference to an alternate movie selected by a 'rmda' (IV-2577) atom.

```
struct ReferenceMovieDataRefRecord {
    long      flags;
    OSType    dataRefType;
    long      dataRefSize;
    char      dataRef[1];
};
```

flags

If set to 0, the data for the 'rmda' atom is imported by reference and the dataRefType, dataRefSize, and dataRef fields are significant. If set to kDataRefIsSelfContained (see below), the alternate movie is self-contained.

dataRefType

The type of dataRef; see “Data References” (IV-2661).

dataRefSize

The size in bytes of dataRef.

dataRef

The data reference. It is typically an `AliasHandle` or a URL (formatted as a C string).

flags Constants

kDataRefIsSelfContained

The reference movie for the 'rmda' atom is self-contained.

Discussion

This structure contains the data for a 'rmda' (IV-2577) atom.

Programming Info

C interface file: MoviesFormat.h

See Also

See 'rmda' (IV-2577).

RegisteredComponentInstanceRecord

Undocumented

```
struct RegisteredComponentInstanceRecord {
    long    data[1];
};
```

data

An array of data.

Programming Info

C interface file: Components.h

RegisteredComponentRecord

Undocumented

```
struct RegisteredComponentRecord {
    long    data[1];
};
```

data

An array of data.

Programming Info

C interface file: Components.h

ResolvedQTEventSpec

Passes data to an MCActionFilterProc (III–2096) callback, when the action mcActionExecuteAllActionsForQTEvent is executed.

```
struct ResolvedQTEventSpec {
    QTAtomSpec    actionAtom;
    Track         targetTrack;
    long          targetRefCon;
};
```

actionAtom

The atom being executed.

ResourceSpec

targetTrack

The affected track.

targetRefCon

A target ID. For VR actions, it's the ID of the hot spot (if any) that triggered the event. For **sprite** tracks, it's the sprite ID of the sprite (if any) that triggered the event.

Programming Info

C interface file: `Movies.h`

ResourceSpec

Describes the **resource** type and resource ID of a component's code, name, information string, or icon.

```
struct ResourceSpec {
    OSType    resType;
    short     resID;
};
```

resType

A 4-byte code (see below) that defines the **resource** type.

resID

The ID of the specific **resource**.

resType Constants

'CODE'

A code **resource**.

'ICON'

An icon **resource**.

'STR '

A name or information string **resource**.

Programming Info

C interface file: `Components.h`

See Also

See the “Component Manager” chapter of *Inside Macintosh: More Macintosh Toolbox* (listed in the **bibliography**).

RGBColor

Defines a color in the red-green-blue system.

```
struct RGBColor {
    unsigned short    red;
    unsigned short    green;
    unsigned short    blue;
};
```

red

The magnitude of the red component

green

The magnitude of the green component

blue

The magnitude of the blue component

Associated Functions

GraphicsImportGetGraphicsMode (I-635)

Programming Info

C interface file: Quickdraw.h

RGBRangeRecord

Defines a range of RGB colors.

```
struct RGBRangeRecord {
    RGBColor    minColor;
    RGBColor    maxColor;
};
```

minColor

One end of the range.

maxColor

The other end of the range.

Programming Info

C interface file: ImageCodec.h

RoutineDescriptor

RoutineDescriptor

Used by the Mac OS Mixed Mode manager to execute a routine.

```
struct RoutineDescriptor {
    UInt16      goMixedModeTrap;
    SInt8       version;
    RDFlagsType routineDescriptorFlags;
    UInt32      reserved1;
    UInt8       reserved2;
    UInt8       selectorInfo;
    UInt16      routineCount;
    RoutineRecord routineRecords[1];
};
```

goMixedModeTrap

Contains 0xAAFE, a trap instruction that tells the 68LC040 emulator to transfer control to the Mixed Mode Manager.

version

The version number of this structure, currently `kRoutineDescriptorVersion` (value = 7).

routineDescriptorFlags

Constants (see below) that specify whether or not the routine selectors are indexable.

reserved1

Reserved; set to 0.

reserved2

Reserved; set to 0.

selectorInfo

Reserved; set to 0.

routineCount

The index of the final routine record in `routineRecords`. If the routine descriptor describes a single procedure, set this field to 0. If it contains pointers to both 680x0 and PowerPC code, set it to 1.

routineRecords

A zero-based array of `RoutineRecord` (IV-2405) structures.

routineDescriptorFlags Constants

kSelectorsAreNotIndexable

For dispatched routines, the recognized routine selectors are not contiguous.

kSelectorsAreIndexable

For dispatched routines, the recognized routine selectors are contiguous and therefore indexable.

Programming Info

C interface file: MixedMode.h

See Also

See RoutineRecord (IV–2405). For information about the Mixed Mode Manager, see *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

RoutineRecord

Contains information about a particular routine in the Mac OS Mixed Mode Manager.

```
struct RoutineRecord {
    ProcInfoType    procInfo;
    SInt8           reserved1;
    ISAType         ISA;
    RoutineFlagsType routineFlags;
    ProcPtr         procDescriptor;
    UInt32          reserved2;
    UInt32          selector;
};
```

procInfo

A constant that defines the routine’s calling conventions and parameters. This constant is controlled by the Mixed Mode Manager; for information about its structure, see *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

reserved1

Reserved; set to 0.

ISA

A constant (see below) that defines the instruction set.

routineFlags

A constant (see below) that specifies information about the routine, or 0x00 if all characteristics are normal.

procDescriptor

A pointer to the routine’s code.

RoutineRecord

reserved2

Reserved; set to 0.

selector

Reserved; set to 0. For routines that are dispatched, this field contains the routine selector.

ISA Constants

kM68kISA

The routine consists of 680x0 code.

kPowerPCISA

The routine consists of PowerPC code.

routineFlags Constants

kProcDescriptorIsRelative

The address of the routine's entry point specified in the `procDescriptor` field is relative to the beginning of the `RoutineDescriptor` (IV-2404) structure. Otherwise it is an absolute address. Value is 0x01.

kFragmentNeedsPreparing

The fragment containing the code to be executed needs to be loaded into memory and prepared by the Code Fragment Manager; otherwise it is already loaded and prepared. If this flag is set, the `kPowerPCISA` and `kProcDescriptorIsRelative` flags should be set. Value is 0x02.

kUseNativeISA

Use the native instruction set architecture instead of the current one. Value is 0x04.

kDontPassSelector

Do not pass the routine selector to the target routine as a parameter. You should not set this flag with 680x0 routines. Value is 0x08.

kRoutineIsDispatchedDefaultRoutine

This routine is the default routine for a set of routines that is dispatched using a routine selector. Value is 0x10.

Programming Info

C interface file: `MixedMode.h`

See Also

See `RoutineDescriptor` (IV-2404). For information about the Mixed Mode Manager, see *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

RTPMPPayloadTypeParams

Describes a streaming payload.

```
struct RTPMPPayloadTypeParams {
    UInt32    flags;
    UInt32    payloadNumber;
    short     nameLength;
    char *    payloadName;
};
```

flags

Flags (see below) that tell whether the payload is static or dynamic. If this field is `kRTPMPPayloadTypeStaticFlag`, then the `payloadNumber` field contains the RTP payload number; if this field is `kRTPMPPayloadTypeDynamicFlag`, then the `payloadName` field contains the name of the dynamic payload encoding being used.

payloadNumber

The RTP payload number.

nameLength

Enter the size of the `payloadName` buffer, counting a null terminator. If the size is too small, QuickTime will return the needed length in this field and will return `paramErr` in the calling function.

payloadName

Pointer to the name of the dynamic payload encoding being used.

flags Constants

`kRTPMPPayloadTypeStaticFlag`

Static payload. Value is 0x00000001.

`kRTPMPPayloadTypeDynamicFlag`

Dynamic payload. Value is 0x00000002.

Programming Info

C interface file: `QTStreamingComponents.h`

RTPMPSampleDataParams

Holds media packetizer sample data, including any number of samples or a partial sample.

```
struct RTPMPSampleDataParams {
    UInt32    version;
```

RTPMPSampleDataParams

```
    UInt32          timeStamp;
    UInt32          duration;
    UInt32          playOffset;
    Fixed           playRate;
    SInt32          flags;
    UInt32          sampleDescSeed;
    Handle          sampleDescription;
    RTPMPSampleRef  sampleRef;
    UInt32          dataLength;
    const UInt8 *   data;
    RTPMPDataReleaseUPP releaseProc;
    void *          refCon;
};
```

version

Version of the data structure. Currently always 0.

timeStamp

RTP time stamp for the presentation of the sample data. This time stamp has already been adjusted by edits, edit rates, etc.

duration

Duration (in RTP time scale) of the sample. For unknown duration, enter 0.

playOffset

Offset within the media sample itself. This is only used for media formats where a single media sample can span across multiple time units. QuickTime Music is an example of this, where a single sample spans the entire track. For most video and audio formats, this will be 0.

playRate

1.0 (0x00010000) is normal. Higher numbers indicate faster play rates. Note that timeStamp is already adjusted by the rate. This field is generally of interest only to audio packetizers.

flags

Flag (see below) to indicate if the sample is a **sync sample** (key frame).

sampleDescSeed

If the sample description changes, this number will change.

sampleDescription

The sample description for the given media sample.

sampleRef

Reserved; do not use.

dataLength

Size of the media data.

data

Pointer to the media data.

releaseProc

If not NIL, you need to call your RTPMPDataReleaseProc (III–2132) when you are finished with the sample data.

refCon

Information to pass to the RTPMPDataReleaseProc.

flags Constants

kRTPMPSyncSampleFlag

The sample is a **sync sample**.

Associated Functions

RTPPBAddPacketSampleData (III–1485)

Programming Info

C interface file: QTStreamingComponents.h

See Also

See RTPMPDataReleaseProc (III–2132).

RTPPayloadCharacteristic

Stores payload characteristics information.

```
struct RTPPayloadCharacteristic {
    OSType    tag;
    long      value;
};
```

tag

Undocumented

value

Undocumented

Programming Info

C interface file: QTStreamingComponents.h

RTPPayloadInfo

Holds a payload's name.

```
struct RTPPayloadInfo {
```

RTPPayloadSortRequest

```
    long    payloadFlags;  
    UInt8   payloadID;  
    char    unused[3];  
    char    payloadName[1];  
};
```

payloadFlags
Undocumented

payloadID
Undocumented

unused[3]
Unused.

payloadName
A name string.

Programming Info

C interface file: QTStreamingComponents.h

RTPPayloadSortRequest

Specifies the sort order for a list of packetizers.

```
struct RTPPayloadSortRequest {  
    long                characteristicCount;  
    RTPPayloadCharacteristic  characteristic[1];  
};
```

characteristicCount
The number of structures in the characteristic field.

characteristic
An array of RTPPayloadCharacteristic (IV–2409) structures.

Associated Functions

QTSFindMediaPacketizer (II–1231)

Programming Info

C interface file: QTStreamingComponents.h

RTPReassemblerInfo

Undocumented


```
struct RTPReassemblerInfo {
    long                characteristicCount;
    RTPPayloadCharacteristic characteristic[1];
```

characteristicCount

The number of structures in the characteristic field.

characteristic

An array of RTPPayloadCharacteristic (IV–2409) structures.

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmInitParams

Initializes a packet reassembler component.

```
struct RTPRssmInitParams {
    RTPSSRC      ssrc;
    UInt8        payloadType;
    UInt8        pad[3];
    TimeBase      timeBase;
    TimeScale     timeScale;
};
```

ssrc

Undocumented

payloadType

Undocumented

pad

Unused.

timeBase

A reference to the reassembler's time base. You obtain a time base by calling GetMovieTimeBase (I–496) or NewTimeBase (II–1119).

timeScale

The reassembler's time scale.

Associated Functions

RTPRssmInitialize (III–1514)

Programming Info

C interface file: QTStreamingComponents.h

RTPRssmPacket

RTPRssmPacket

A streaming reassembler packet list.

```
struct RTPRssmPacket {
    struct RTPRssmPacket *    next;
    struct RTPRssmPacket *    prev;
    QTSSStreamBuffer *        streamBuffer;
    Boolean                    paramsFilledIn;
    UInt8                      pad[1];
    UInt16                     sequenceNum;
    UInt32                     transportHeaderLength;
    UInt32                     payloadHeaderLength;
    UInt32                     dataLength;
    SHServerEditParameters    serverEditParams;
    TimeValue64                timeStamp;
    SInt32                     chunkFlags;
    SInt32                     flags;
};
```

next

A pointer to the next RTPRssmPacket structure.

prev

A pointer to the previous RTPRssmPacket structure.

streamBuffer

A pointer to a QTSSStreamBuffer (IV-2385) structure defining the stream buffer.

paramsFilledIn

Undocumented

pad

Undocumented

sequenceNum

Undocumented

transportHeaderLength

Undocumented

payloadHeaderLength

Undocumented

dataLength

Undocumented

serverEditParams

Undocumented

timeStamp
Undocumented
chunkFlags
Undocumented
flags
Undocumented

Associated Functions
RTPRssmAdjustPacketParams (III–1500)

Programming Info
C interface file: QTStreamingComponents.h

RTPSendStreamBufferRangeParams

Undocumented

```
struct RTPSendStreamBufferRangeParams {
    QTSSStreamBuffer *      streamBuffer;
    TimeValue64             presentationTime;
    UInt32                  chunkStartPosition;
    UInt32                  numDataBytes;
    SInt32                  chunkFlags;
    SInt32                  flags;
    const SHServerEditParameters * serverEditParams;
};
```

streamBuffer
Undocumented
presentationTime
Undocumented
chunkStartPosition
Undocumented
numDataBytes
Undocumented
chunkFlags
Undocumented
flags
Undocumented

SampleDescription

serverEditParams
Undocumented

Associated Functions

RTPRssmSendStreamBufferRange (III–1519)

Programming Info

C interface file: QTStreamingComponents.h

SampleDescription

Stores information required to decode samples in a media.

```
struct SampleDescription {  
    long    descSize;  
    long    dataFormat;  
    long    resvd1;  
    short   resvd2;  
    short   dataRefIndex;  
};
```

descSize
Size in bytes of this data structure.

dataFormat
The sample description format.

resvd1
Reserved. Set to 0.

resvd2
Reserved. Set to 0.

dataRefIndex
Contains an index value indicating which of the media's data references contains the sample data for this sample description.

Associated Functions

AddMediaSample (I–27)

Programming Info

C interface file: Movies.h

See Also

For an atom that contains sample description structures, see 'stds' (IV–2591).

SampleReference64Record

Provides a 64-bit version of SampleReferenceRecord (IV–2416).

```
struct SampleReference64Record {
    wide           dataOffset;
    unsigned long  dataSize;
    TimeValue      durationPerSample;
    unsigned long  numberOfSamples;
    short          sampleFlags;
};
```

dataOffset

Specifies the offset into the movie data file. This field specifies the offset into the file of the sample data.

dataSize

Specifies the total number of bytes of sample data identified by the reference. All samples referenced by a single SampleReference64Record must be the same size.

durationPerSample

Specifies the duration of each sample in the reference. You must specify this parameter in the media's time scale. All samples referenced by a single SampleReference64Record must be the same duration.

numberOfSamples

Specifies the number of samples contained in the reference.

sampleFlag

Contains flags (see below) that control the operation. Set unused flags to 0.

sampleFlag Constants

mediaSampleNotSync

Indicates whether the sample to be added is not a synchronous sample. Set this flag to 1 if the sample is not a synchronous sample. Set this flag to 0 if the sample is a synchronous sample.

Programming Info

C interface file: `Movies.h`

See Also

See SampleReferenceRecord (IV–2416).

SampleReferenceRecord

SampleReferenceRecord

Describes a sample or group of similar samples.

```
struct SampleReferenceRecord {  
    long        dataOffset;  
    long        dataSize;  
    TimeValue   durationPerSample;  
    long        numberOfSamples;  
    short       sampleFlags;  
};
```

`dataOffset`

Specifies the offset into the movie data file. This field specifies the offset into the file of the sample data.

`dataSize`

Specifies the total number of bytes of sample data identified by the reference. All samples referenced by a single `SampleReferenceRecord` must be the same size.

`durationPerSample`

Specifies the duration of each sample in the reference. You must specify this parameter in the media's time scale. All samples referenced by a single `SampleReferenceRecord` must be the same duration.

`numberOfSamples`

Specifies the number of samples contained in the reference.

`sampleFlag`

Contains flags (see below) that control the operation. Set unused flags to 0.

sampleFlag Constants

`mediaSampleNotSync`

Indicates whether the sample to be added is not a synchronous sample. Set this flag to 1 if the sample is not a synchronous sample. Set this flag to 0 if the sample is a synchronous sample.

Programming Info

C interface file: `Movies.h`

See Also

This data structure is used by `GetMediaSampleReferences` (I-449) and `AddMediaSampleReferences` (I-33).

SampleToChunk

Maps a physical sample number to a chunk number.

```
struct SampleToChunk {
    long    firstChunk;
    long    samplesPerChunk;
    long    sampleDescriptionID;
};
```

firstChunk

The first chunk number that uses this data structure.

samplesPerChunk

The number of samples in each chunk.

sampleDescriptionID

An index into the 'stds' (IV-2591) atom for this sample.

Discussion

A chunk is a collection of data samples in a media that allows optimized data access. A chunk may contain one or more samples. Chunks in a media may have different sizes, and the samples within a chunk may have different sizes.

Programming Info

C interface file: `MoviesFormat.h`

See Also

See 'stds' (IV-2591).

SCDataRateSettings

Contains the current temporal compression parameters that govern the data rate.

```
struct SCDataRateSettings {
    long    dataRate;
    long    frameDuration;
    CodecQ  minSpatialQuality;
    CodecQ  minTemporalQuality;
};
```

dataRate

Specifies the maximum number of bytes of compressed data your application wants to receive per second. Use this parameter to modulate the rate at which

SCExtendedProcs

the component passes compressed data to your application. This can be useful to account for hardware limitations during sequence playback.

frameDuration

Indicates the duration of each frame, in milliseconds. Set this parameter to 0 to allow the standard dialog component to calculate the duration based upon the frame rate you specify in an `scTemporalSettingsType` request. However, if you allow the user to specify a 0 frame rate (that is, you set the `scAllowZeroFrameRate` flag to 1 in your `scPreferenceFlagsType` request), you must set the frame duration each time you compress a frame, because the component does not have sufficient information to determine an appropriate rate.

minSpatialQuality

Specifies the minimum acceptable spatial quality. In order to meet your specified data rate, the standard dialog component may have to adjust the spatial quality setting. Use this parameter to set a minimum level, which the component may not exceed. See the chapter “Image Compression Manager” in *Inside Macintosh: QuickTime* (listed in the **bibliography**) for values for both this parameter and the `minTemporalQuality` parameter.

minTemporalQuality

Specifies the minimum acceptable temporal quality. As with spatial quality, in order to meet your specified data rate, the standard dialog component may have to adjust the temporal quality setting. Use this parameter to set a minimum level, which the component may not exceed.

Programming Info

C interface file: `QuickTimeComponents.h`

SCExtendedProcs

Extends the interface provided in the standard image or sequence dialog boxes.

```
struct SCExtendedProcs {
    SCModalFilterUPP    filterProc;
    SCModalHookUPP      hookProc;
    long                refcon;
    Str31                customName;
};
```

filterProc

Contains a pointer to an `SCModalFilterProc` (III–2133) callback in your application. Because the compression dialog box is a movable modal dialog

box, you must provide a filter to process update events for your application windows. The standard component calls your filter function before it processes the event. You can use this function to control events in the dialog box. For example, you might use the filter function to release processing time to other windows displayed by your application while the standard image-compression dialog box is being displayed. If you do not want to specify a filter function, set this parameter to NIL.

hookProc

Contains a pointer to an `SCModalHookProc` (III–2134) callback in your application. The standard component calls your hook function whenever the user selects an item in the dialog box. You can use this function to customize the operation of the standard image-compression dialog box. For example, you might want to support a custom button that activates a secondary dialog box. Another possibility would be to provide additional validation support when the user clicks OK. If you do not want to specify a hook function, set this parameter to NIL.

refcon

Specifies a reference constant that is to be passed to the `SCModalHookProc` and `SCModalFilterProc` callbacks.

customName

Specifies the string to be displayed in the custom button in the dialog box. If you are not using a custom button, set this parameter to NIL.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `SCModalFilterProc` (III–2133) and `SCModalHookProc` (III–2134). The operation of modal-dialog filter functions is described in the chapter “Dialog Manager” in *Inside Macintosh: Macintosh Toolbox Essentials* (listed in the **bibliography**).

ScheduledSoundHeader

Provides data to the `scheduledSoundCmd` command for `SndDoImmediate` (III–1832) and `SndDoCommand` (III–1831).

```
struct ScheduledSoundHeader {
    SoundHeaderUnion    u;
    long                flags;
    short               reserved;
    short               callBackParam1;
    long                callBackParam2;
```

SCParams

```
    TimeRecord      startTime;
};
```

u

A union of one of the three existing SoundHeader types.

flags

Flag (see below) to determine if this structure is an
ExtendedScheduledSoundHeader.

reserved

Reserved.

callbackParam1

Undocumented

callbackParam2

Undocumented

startTime

Undocumented

flags Constant

kScheduledSoundExtendedHdr

If 0, this is a classic ScheduledSoundHeader. If 1, this is an
ExtendedScheduledSoundHeader (IV–2251) and at least contains the recordSize
and extendedFlags fields.

Programming Info

C interface file: Sound.h

See Also

See SndDoImmediate (III–1832) and SndDoCommand (III–1831).

SCParams

Provides data for the SCGetCompressionExtended (III–1544) function.

```
struct SCParams {
    long          flags;
    CodecType     theCodecType;
    CodecComponent theCodec;
    CodecQ        spatialQuality;
    CodecQ        temporalQuality;
    short         depth;
    Fixed         frameRate;
    long          keyFrameRate;
    long          reserved1;
```

```
    long                reserved2;
};
```

flags

Flags (see below).

theCodecType

A compressor type; see “Codec Identifiers” (IV–2655).

theCodec

An instance of a compressor component, obtained by calling `OpenComponent` (II–1130) or `OpenDefaultComponent` (II–1131).

spatialQuality

Constants (see below) that determine image spatial quality.

temporalQuality

Constants (see below) that determine image temporal quality.

depth

Image data depth.

frameRate

The frame rate.

keyFrameRate

The **key frame** rate.

reserved1

Reserved.

reserved2

Reserved.

flags Constants

scGetCompression

Undocumented

scShowMotionSettings

Undocumented

scSettingsChangedItem

Undocumented

spatialQuality and temporalQuality Constants

codecMinQuality

The minimum valid value for a CodecQ field.

codecLowQuality

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

ScrpSTElement

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value for a CodecQ field.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Associated Functions

`SCGetCompressionExtended` (III–1544)

Programming Info

C interface file: `QuickTimeComponents.h`

ScrpSTElement

Defines the style of text in the Mac OS scrap.

```
struct ScrpSTElement {
    long        scrpStartChar;
    short       scrpHeight;
    short       scrpAscent;
    short       scrpFont;
    StyleField  scrpFace;
    short       scrpSize;
    RGBColor    scrpColor;
};
```

`scrpStartChar`

The offset to the beginning of a `TEStylerec` (IV–2473) structure in the Mac OS scrap.

`scrpHeight`

The line height.

`scrpAscent`

The font ascent.

`scrpFont`

The font family ID.

scrpFace

Unpacked byte representing the style.

scrpSize

The size in points.

scrpColor

An RGBColor (IV–2403) structure representing the text color.

Programming Info

C interface file: `Movies.h`

SCSpatialSettings

Passes information when `scSpatialSettingsType` is used with `SCGetInfo` (III–1545) or `SCSetInfo` (III–1558).

```
struct SCSpatialSettings {
    CodecType      codecType;
    CodecComponent codec;
    short          depth;
    CodecQ         spatialQuality;
};
```

`codecType`

Specifies the default compressor type that is displayed in the pop-up menu of compressors in the dialog box. The standard image-compression dialog component uses this field to return the compressor type that was selected by the user. You must set this parameter to one of the compressor types supported by the Image Compression Manager, or to `NIL`. If you set the field to `NIL`, the standard image-compression dialog component uses as the default value the first compressor or compressor type that it retrieves from the Image Compression Manager.

`codec`

Provides additional information about the default compressor that is displayed in the pop-up menu of compressors in the dialog box. If the user selects a specific compressor component, the standard image-compression dialog component returns the appropriate compressor identifier in this field.

The `scListEveryCodec` bit in the `scPreferenceFlagsType` request passed to `SCGetInfo` or `SCSetInfo` influences the operation of the compressor list in the dialog box and, therefore, the way the component uses this field. Set `scListEveryCodec` to 1 to have the list contain an entry for each compressor

component in the system. If the flag is set to 1, the standard image-compression dialog component uses this field along with the `codecType` field to select the default compressor that appears in the dialog box. To specify a default image compressor component, set the `codecType` field to the appropriate compressor identifier. When the user clicks OK in the dialog box, the standard image-compression dialog component returns the compressor identifier that corresponds to the selected image compressor component. If you set the `codecType` field to NIL, the standard image-compression dialog component uses as the default value the first compressor of the specified type that it retrieves from the Image Compression Manager.

If you have set `scListEveryCodec` to 0, the list contains only one entry for each type of compressor in the system. The standard image-compression dialog component ignores this field when creating the list of compressor types. In this case, the standard image-compression dialog component does not change the value of this field when the user clicks OK.

However, you may use the `codec` field to specify additional selection criteria by setting it to one of the special compressor identifiers supported by the Image Compression Manager; see “Codec Identifiers” (IV–2655). The standard image-compression dialog component may use this value when it validates the compression parameters selected by the user.

`depth`

Specifies the default value of the pixel depth pop-up menu in the dialog box. This menu allows the user to select the color or gray scale resolution value to be used when compressing the image or image sequence. If you set this field to 0, the component chooses an appropriate depth for the default compressor you specified with the `codec` field. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the depth of color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

When the user clicks OK, the standard image-compression dialog component sets this field to the pixel depth value selected by the user. Note that the standard image-compression dialog component may adjust the depth value so that it corresponds to a value that is supported by the compressor that has been selected by the user. The depth returned could be 0 if the `scShowBestDepth` flag is set in `scPreferenceFlagsType`.

`spatialQuality`

A constant (see below) that specifies the default setting of the quality slider in the dialog box. This slider controls the spatial quality of the compressed image

sequence, which influences the amount of spatial compression that can be achieved. Spatial compression eliminates redundant information within each frame in a sequence. When the user clicks OK, the standard image-compression dialog component sets this field to the spatial quality value selected by the user. Note that the standard image-compression dialog component may adjust the quality value so that it corresponds to a value that is supported by the compressor that has been selected by the user.

spatialQuality Constants

`codecMinQuality`

The minimum valid value.

`codecLowQuality`

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

This structure controls how an image is compressed; you supply a pointer to it. If you are retrieving its settings, using `SCGetInfo` (III–1545), the Standard Dialog component places the current settings into the structure. If you are changing the settings, using `SCSetInfo` (III–1558), place the new values into the structure; the Standard Dialog component will use those values to update its settings.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `SCGetInfo` (III–1545) and `SCSetInfo` (III–1558).

SCStatus

Passes sound channel status information to `SndChannelStatus` (III–1829).

SCStatus

```
struct SCStatus {
    UnsignedFixed    scStartTime;
    UnsignedFixed    scEndTime;
    UnsignedFixed    scCurrentTime;
    Boolean          scChannelBusy;
    Boolean          scChannelDisposed;
    Boolean          scChannelPaused;
    Boolean          scUnused;
    unsigned long    scChannelAttributes;
    long             scCPULoad;
};
```

scStartTime

Starting time in seconds for play from disk; 0 if the Sound Manager is not currently playing from a disk through the channel specified to `SndChannelStatus`. The Sound Manager sets the values of the `scStartTime` and `scEndTime` fields based on the values you set in an `AudioSelection` (IV-2161) structure.

scEndTime

Ending time in seconds for play from disk; 0 if the Sound Manager is not currently playing from a disk through the channel specified to `SndChannelStatus`.

scCurrentTime

Current time for play from disk; 0 if the Sound Manager is not currently playing from a disk through the channel specified to `SndChannelStatus`. This field indicates the number of seconds between the beginning of the sound on the disk and the part of the sound currently being played.

scChannelBusy

TRUE if the channel is processing commands; FALSE otherwise.

scChannelDisposed

Reserved; do not use.

scChannelPaused

TRUE if the channel is paused; FALSE otherwise. After issuing a series of sound commands, you can use the `scChannelBusy` and `scChannelPaused` fields to determine if the channel has finished processing all of the commands. If both `scChannelBusy` and `scChannelPaused` are FALSE, the Sound Manager has processed all of the channel's commands.

scUnused

Unused.

scChannelAttributes

You can use constants (see below) to mask out certain values in the scChannelAttributes field to determine how a channel has been initialized.

scCPULoad

Obsolete; do not use.

scChannelAttributes Mask Constants

initPanMask

Mask for right/left pan values; value is 0x0003.

initSRateMask

Mask for sample rate values; value is 0x0030.

initStereoMask

Mask for mono/stereo values; value is 0x00C0.

Discussion

Note that because the Sound Manager might be playing only a selection of a sound, the scCurrentTime field does not reflect the number of seconds of sound play that have elapsed. To compute the number of seconds of sound play elapsed, you can subtract the value in the scStartTime field from that in the scCurrentTime field. However, because the Sound Manager updates the value of the scCurrentTime field only periodically, you should not rely on the accuracy of its value.

Version Notes

The scCPULoad field previously reflected the percentage of CPU processing power used by the sound channel. However, this field is obsolete, and you should not rely on its value.

Associated Functions

SndChannelStatus (III–1829)

Programming Info

C interface file: Sound.h

See Also

See SndChannelStatus (III–1829).

SCTemporalSettings

Passes information when scTemporalSettingsType is used with SCGetInfo (III–1545) or SCSetInfo (III–1558).

```
struct SCTemporalSettings {
    CodecQ    temporalQuality;
    Fixed     frameRate;
```

SCTemporalSettings

```
long    keyFrameRate;  
};
```

temporalQuality

A constant (see below) that specifies the default setting of the motion quality slider in the dialog box. This slider controls the temporal quality of the compressed image, which influences the amount of temporal compression that can be achieved. Note that Apple's component uses the same slider for both spatial and temporal quality. Temporal compression eliminates redundant information between frames in an image sequence. When the user clicks OK, the standard image-compression dialog component sets this field to the temporal quality value selected by the user. Note that the standard image-compression dialog component may adjust the quality value so that it corresponds to a value that is supported by the compressor that has been selected by the user.

frameRate

Specifies the default value of the text-edit box that controls the number of frames per second in the image sequence to be compressed. This dialog item allows the user to select the frame rate to be used when compressing the image sequence. Note that this field is stored as a fixed-point number, allowing the user to specify fractional frame rates. When the user clicks OK, the standard image-compression dialog component sets this field to the frame rate value specified by the user. If you have set the `scAllowZeroFrameRate` flag to 1 in the `scPreferenceFlagsType` request passed to `SCGetInfo` or `SCSetInfo`, and the user specifies nothing or 0, the component sets this field to 0.

keyFrameRate

Specifies the default value of the text-edit box that controls the frequency with which **key frames** are inserted into the compressed image sequence. Key frames provide points from which a temporally compressed sequence may be decompressed. When the user clicks OK, the standard image-compression dialog component sets this field to the key frame rate value specified by the user. If you have set the `scAllowZeroKeyFrameRate` flag to 1 in the `scPreferenceFlagsType` request passed to `SCGetInfo` or `SCSetInfo`, and the user specifies nothing or 0, the component sets this field to 0.

temporalQuality Constants

codecMinQuality

The minimum valid value.

codecLowQuality

Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Image reproduction of normal quality.

`codecHighQuality`

High-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`

The maximum standard value.

`codecLosslessQuality`

Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Discussion

The frame rate dialog item can be useful in cases where your application cannot determine the frame rate of the source movie. For example, movies stored in PICT files do not include frame rate information. Therefore, the user must specify a frame rate for you. Alternatively, some users may want to create movies with different frame rates. This item allows the user to specify a rate for the compressed sequence.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `SCGetInfo` (III–1545) and `SCSetInfo` (III–1558).

SeqGrabExtendedFrameInfo

Defines a frame for a sequence grabber component and its sequence grabber channel components.

```
struct SeqGrabExtendedFrameInfo {
    wide      frameOffset;
    long      frameTime;
    long      frameSize;
    SGChannel  frameChannel;
    long      frameRefCon;
    SGOutput  frameOutput;
};
```

`frameOffset`

Specifies the offset to the sample. Note that this is a 64-bit value.

SeqGrabFrameInfo

`frameTime`

Specifies the time at which the frame was captured by a sequence grabber channel component. The time value is relative to the data sequence. The channel component must choose a time scale and use it consistently for all sample references.

`frameSize`

Specifies the number of bytes in the current sample.

`frameChannel`

Identifies the current connection to the channel component.

`frameRefCon`

Contains a reference constant for use by the channel component. The channel component uses this constant in any appropriate way; for example, to store a reference to frame differencing information for a time-compressed sequence.

`frameOutput`

Identifies the sequence grabber output used to store captured data referenced by the current record.

Discussion

This structure differs from `SeqGrabFrameInfo` (IV–2430) in two respects: the `frameOffset` field takes a 64-bit value, and the `frameOutput` field does not exist in `SeqGrabFrameInfo`.

Version Notes

Introduced in QuickTime 4.

Associated Functions

`SGAddExtendedFrameReference` (III–1677)

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `SeqGrabFrameInfo` (IV–2430).

SeqGrabFrameInfo

Provides information about a frame for a sequence grabber component and its sequence grabber channel components.

```
struct SeqGrabFrameInfo {
    long        frameOffset;
    long        frameTime;
    long        frameSize;
```

```

        SGChannel    frameChannel;
        long         frameRefCon;
};

```

frameOffset

Specifies the offset to the sample. Your channel component obtains this value from `SGWriteMovieData` (III–1824).

frameTime

Specifies the time at which your channel component captured the frame. This time value is relative to the data sequence. That is, this time is not represented in the context of any fixed time scale. Rather, your channel component must choose and use a time scale consistently for all sample references.

frameSize

Specifies the number of bytes in the sample described by the sample reference.

frameChannel

Identifies the current connection to your channel.

frameRefCon

Contains a reference constant for use by your channel component. You can use this value in any way that is appropriate for your channel component. For example, video channel components may use this value to store a reference to frame differencing information for a temporally compressed image sequence.

Discussion

This structure differs from `SeqGrabExtendedFrameInfo` (IV–2429) in two respects: the `frameOffset` field takes a 32-bit value, and `SeqGrabExtendedFrameInfo` has a `frameOutput` field.

Associated Functions

`SGAddFrameReference` (III–1681)

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `SeqGrabExtendedFrameInfo` (IV–2429).

SFReply

Passes Standard File Dialog information to `SFGetFilePreview` (III–1674) or `SFPGetFilePreview` (III–1676).

```

struct SFReply {

```

SGCompressInfo

```
Boolean      good;  
Boolean      copy;  
OSType       fType;  
short        vRefNum;  
short        version;  
StrFileName  fName;  
};
```

good

Reports whether the reply record is valid. The value is `TRUE` after the user clicks Save or Open; `FALSE` after the user clicks Cancel. When the user has completed the dialog box, the other fields in the reply record are valid only if the value of good is `TRUE`.

copy

Reserved; do not use.

fType

Contains the file type of the selected file; see “File Types and Creators” (IV–2668).

vRefNum

Contains the working directory reference number, which encodes both the volume reference number and the parent directory ID of the selected file.

version

Reserved; do not use.

fName

Contains the name of the selected file; its type is `Str63` on the Mac OS.

Associated Functions

`SFGetFilePreview` (III–1674)

Programming Info

C interface file: `StandardFile.h`

See Also

See `SFGetFilePreview` (III–1674) and `SFPGetFilePreview` (III–1676).

SGCompressInfo

Defines the characteristics of a buffer that contains a captured image that has been compressed.

```
struct SGCompressInfo {  
    Ptr          buffer;
```

```

        unsigned long    bufferSize;
        UInt8            similarity;
        UInt8            reserved;
};

```

buffer

Points to the buffer that contains the compressed image. This pointer must contain a **32-bit clean** address.

bufferSize

Specifies the number of bytes of image data in the buffer.

similarity

Indicates the relative similarity of this image to the previous image in a sequence. A value of 0 indicates that the current frame is a **key frame** in the sequence. A value of 255 indicates that the current frame is identical to the previous frame. Values from 1 through 254 indicate relative similarity, ranging from very different (1) to very similar (254).

reserved

Reserved; set to 0.

Discussion

Callback functions use this structure to exchange information about compressed images. For example, `SGCompressCompleteBottleProc` (III–2138) must format a compression information record whenever a video frame is compressed.

Associated Functions

`SGCompressCompleteBottleProc` (III–2138)

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `SGCompressCompleteBottleProc` (III–2138).

SGDeviceListRecord

Passes information about one or more channel devices.

```

struct SGDeviceListRecord {
    short        count;
    short        selectedIndex;
    long         reserved;
    SGDeviceName entry[1];
};

```

SGDeviceName

count

Indicates the number of devices described by this structure. The value of this field corresponds to the number of entries in the device name array defined by the entry field.

selectedIndex

Identifies the currently active device. The value of this field corresponds to the appropriate entry in the device name array defined by the entry field. Note that this value is zero-relative; that is, the first entry has an index number of 0, the second's value is 1, and so on.

reserved

Reserved; set to 0.

entry

Contains an array of device name structures. Each structure corresponds to one valid device. The count field indicates the number of entries in this array. The SGDeviceName (IV–2434) structure defines the format of a device name structure.

Discussion

Sequence grabbers provide a number of functions that allow you to determine the device that is attached to a given sequence grabber channel. These devices allow the channel component to control the digitizing equipment. For example, video channels use video digitizer components, and sound channels use sound input drivers. Your application can use these routines to present a list of available devices to the user, allowing the user to select a specific device for each channel.

Programming Info

C interface file: QuickTimeComponents.h

See Also

SGGetChannelDeviceList (III–1701) retrieves a list of devices that may be used with a specified channel. You can place one or more device names into a menu by calling SGAppendDeviceListToMenu (III–1685). You can use SGSetChannelDevice (III–1776) to assign a device to a channel.

SGDeviceName

Identifies a single device that may be used by a sequence grabber channel.

```
struct SGDeviceName {
    Str63    name;
    Handle   icon;
    long     flags;
    long     refCon;
```



```

    long    reserved;
};

```

name

Contains the name of the device. For video digitizer components, this field contains the component's name as specified in the component **resource**. For sound input drivers, this field contains the driver name.

icon

Contains a handle to the device's icon. Some devices may support an icon, which you may choose to present to the user. If the device does not support an icon, or if you choose not to retrieve this information, by setting the `sgDeviceListWithIcons` flag to 0 when you call `SGGetChannelDeviceList` (III-1701), this field is set to NIL.

flags

Flags (see below) that reflect the current status of the device. The sequence grabber sets these flags when you retrieve a device list.

refCon

Reserved; set to 0.

reserved

Reserved; set to 0.

flags Constants

`sgDeviceNameFlagDeviceUnavailable`

When set to 1, this flag indicates that this device is not currently available.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `SGDeviceListRecord` (IV-2433).

SGOutputRecord

Contains a sequence grabber output.

```

struct SGOutputRecord {
    long    data[1];
};

```

data

An array of data.

ShadowSync

Programming Info

C interface file: QuickTimeComponents.h

ShadowSync

Identifies a self-contained **sync sample** that is used as an alternate for an existing frame difference sample.

```
struct ShadowSync {  
    long    fdSampleNum;  
    long    syncSampleNum;  
};
```

fdSampleNum

A frame-difference sample number.

syncSampleNum

The corresponding **sync sample** number.

Programming Info

C interface file: MoviesFormat.h

See Also

For the atom structure that holds ShadowSync data structures, see 'stsh' (IV–2592).

SHChunkRecord

Defines a chunk for a reassembler.

```
struct SHChunkRecord {  
    UInt32          version;  
    long            reserved1;  
    SInt32          flags;  
    UInt32          dataSize;  
    const UInt8 *   dataPtr;  
    long            reserved2;  
    long            reserved3;  
    TimeValue64     presentationTime;  
    long            reserved4;  
    long            reserved5;  
    const SHServerEditParameters * serverEditParameters;  
    long            reserved6;  
    long            reserved7;  
};
```

version

Undocumented

reserved1

Reserved; do not use.

flags

Undocumented

dataSize

The size of the chunk data.

dataPtr

A pointer to the chunk data.

reserved2

Reserved; do not use.

reserved3

Reserved; do not use.

presentationTime

Undocumented

reserved4

Reserved; do not use.

reserved5

Reserved; do not use.

serverEditParameters

A pointer to an SHServerEditParameters (IV–2437) structure containing the server edit parameters

reserved6

Reserved; do not use.

reserved7

Reserved; do not use.

Associated Functions

RTPRssmCopyDataToChunk (III–1503)

Programming Info

C interface file: QTStreamingComponents.h

SHServerEditParameters

Undocumented

SMStatus

```
struct SHServerEditParameters {
    UInt32      version;
    Fixed       editRate;
    TimeValue64 dataStartTime_mediaAxis;
    TimeValue64 dataEndTime_mediaAxis;
};
```

version

Undocumented

editRate

Undocumented

dataStartTime_mediaAxis

Undocumented

dataEndTime_mediaAxis

Undocumented

Programming Info

C interface file: QTStreamingComponents.h

SMStatus

Passes information to SndManagerStatus (III–1839) to determine information about the sound channels currently allocated by all applications.

```
struct SMStatus {
    short    smMaxCPULoad;
    short    smNumChannels;
    short    smCurCPULoad;
};
```

smMaxCPULoad

A default value of 100 at startup time.

smNumChannels

The number of sound channels currently allocated. This does not mean that the channels are actually being used, only that they have been created with the SndNewChannel function and not yet disposed.

smCurCPULoad

The approximate percentage of CPU processing power currently taken by allocated sound channels.

Discussion

The Sound Manager uses information that it returns in the `smMaxCPULoad` and `smCurCPULoad` fields to help it determine whether it can allocate a new channel when your application calls `SndNewChannel` (III–1839). Your application should not reply on the values returned in the `smMaxCPULoad` and `smCurCPULoad` fields. To determine if it is safe to allocate a channel, simply try to allocate it with `SndNewChannel`. That function returns the appropriate result code if allocating the channel would put too much of a strain on CPU processing.

Associated Functions

`SndManagerStatus` (III–1839)

Programming Info

C interface file: `Sound.h`

See Also

See `SndManagerStatus` (III–1839).

Snd2ListResource

HyperCard sound **resource** format.

```
struct Snd2ListResource {
    short      format;
    short      refCount;
    short      numCommands;
    SndCommand commandPart[1];
    UInt8      dataPart[1];
};
```

`format`

Undocumented

`refCount`

Undocumented

`numCommands`

Undocumented

`commandPart`

Undocumented

`dataPart`

Undocumented

Programming Info

C interface file: `Sound.h`

SndChannel

SndChannel

Stores information about each sound channel.

```
struct SndChannel {
    SndChannelPtr    nextChan;
    Ptr              firstMod;
    SndCallbackUPP   callBack;
    long              userInfo;
    long              wait;
    SndCommand        cmdInProgress;
    short             flags;
    short             qLength;
    short             qHead;
    short             qTail;
    SndCommand        queue[128];
};
```

nextChan

A pointer to the next sound channel in a single queue of channels that the Sound Manager maintains for all applications.

firstMod

Reserved; do not use.

callBack

A pointer to an SndCallbackProc (III–2147) callback associated with the sound channel.

userInfo

A four-byte value that your application can use to store information.

wait

Reserved; do not use.

cmdInProgress

Reserved; do not use.

flags

Reserved; do not use.

qLength

Reserved; do not use.

qHead

Reserved; do not use.

qTail

Reserved; do not use.

queue

The sound commands pending for the sound channel.

Discussion

The Sound Manager maintains a structure of this type to store information about each sound channel that you allocate directly by calling the `SndNewChannel` (III–1839) function or indirectly by passing a NIL channel to a high-level Sound Manager routine like `SndPlay` (III–1842). The only field of this structure that you are likely to need to access directly is the `userInfo` field. This field is useful if you need to pass a value to a Sound Manager callback procedure or completion routine. For example, you might store a handle to sound data here so that a routine that disposes of an allocated channel can also release the sound data that the channel played.

Associated Functions

`SndCallbackProc` (III–2147)

Programming Info

C interface file: `Sound.h`

SndCommand

Stores a sound command.

```
struct SndCommand {
    unsigned short  cmd;
    short           param1;
    long            param2;
};
```

`cmd`

The command number; see “Sound Commands” (IV–2691).

`param1`

First parameter.

`param2`

Second parameter.

Discussion

Commands are always 8 bytes in length. The first 2 bytes are the command number, and the next 6 make up the command’s options. The format of the last 6 bytes depends on the command in use, although typically those 6 bytes are interpreted as an integer followed by a long integer. For example, an application can install a wave table into a sound channel by using `SndDoCommand` (III–1831) with a sound command whose `cmd` field is the `waveTableCmd` constant. In that case, the `param1` field specifies

SndDoubleBuffer

the length of the wave table, and the param2 field is a pointer to the wave-table data itself. Other sound commands may interpret the 6 parameter bytes differently or may not use them at all.

Associated Functions

SndCallbackProc (III–2147)

Programming Info

C interface file: Sound.h

SndDoubleBuffer

Buffers between which the Sound Manager switches until all sound data has been sent into the sound channel.

```
struct SndDoubleBuffer {  
    long    dbNumFrames;  
    long    dbFlags;  
    long    dbUserInfo[2];  
    SInt8    dbSoundData[1];  
};
```

dbNumFrames

The number of sound frames in the buffer.

dbFlags

Status flags (see below).

dbUserInfo

A value that your application can use to store information.

dbSoundData

The sound data.

dbFlags Constants

dbBufferReady

The buffer is ready.

dbLastBuffer

This is the last buffer of data.

Associated Functions

SndDoubleBackProc (III–2148)

Programming Info

C interface file: Sound.h

SndDoubleBufferHeader

Passes information to SndPlayDoubleBuffer (III–1843).

```
struct SndDoubleBufferHeader {
    short          dbhNumChannels;
    short          dbhSampleSize;
    short          dbhCompressionID;
    short          dbhPacketSize;
    UnsignedFixed  dbhSampleRate;
    SndDoubleBufferPtr dbhBufferPtr[2];
    SndDoubleBackUPP dbhDoubleBack;
};
```

`dbhNumChannels`

The number of sound channels.

`dbhSampleSize`

The sample size, if sound is noncompressed.

`dbhCompressionID`

The compression algorithm used on the samples in the compressed sound header (see below). The constant `fixedCompression` is available only with Sound Manager versions 3.0 and later. If the `compressionID` field contains the value `fixedCompression`, the Sound Manager reads the `format` field to determine the compression algorithm used to generate the compressed data. Otherwise, the Sound Manager reads the `compressionID` field. Apple reserves the right to use compression IDs in the range 0 through 511. Currently the constant `variableCompression` is not used by the Sound Manager.

`dbhPacketSize`

The size, in bits, of the smallest element that a given expansion algorithm can work with (see below). Beginning with Sound Manager version 3.0, you can specify the value 0 in this field to instruct the Sound Manager to determine the packet size itself.

`dbhSampleRate`

The sample rate.

`dbhBufferPtr`

Pointers to SndDoubleBuffer (IV–2442) structures.

`dbhDoubleBack`

Pointer to an SndDoubleBackProc (III–2148) callback.

SndDoubleBufferHeader2

compressionID Constants

- `variableCompression`
Variable-ratio compression; value is `-2`.
- `fixedCompression`
Fixed-ratio compression; value is `-1`.
- `notCompressed`
Uncompressed samples; value is `0`.
- `threeToOne`
3:1 compressed samples; value is `3`.
- `sixToOne`
6:1 compressed samples; value is `4`.

Discussion

The `dbhBufferPtr` array contains pointers to two structures of type `SndDoubleBuffer` (IV–2442). These are the two buffers between which the Sound Manager switches until all the sound data has been sent into the sound channel. When the call to `SndPlayDoubleBuffer` (III–1843) is made, the two buffers should both already contain a nonzero number of frames of data.

Version Notes

The Sound Manager defines `SndDoubleBufferHeader2` (IV–2444), identical to `SndDoubleBufferHeader` except that it contains the `dbhFormat` field (of type `OSType`) that defines a custom codec to be used to decompress the sound data. The `dbhFormat` field is used only if the `dbhCompressionID` field of `SndDoubleBufferHeader2` contains the value `fixedCompression`.

Associated Functions

`SndPlayDoubleBuffer` (III–1843)

Programming Info

C interface file: `Sound.h`

See Also

See `SndDoubleBufferHeader2` (IV–2444).

SndDoubleBufferHeader2

Passes information to `SndPlayDoubleBuffer` (III–1843), specifying a codec.

```
struct SndDoubleBufferHeader2 {
    short          dbhNumChannels;
    short          dbhSampleSize;
    short          dbhCompressionID;
```

```

short                dbhPacketSize;
UnsignedFixed        dbhSampleRate;
SndDoubleBufferPtr   dbhBufferPtr[2];
SndDoubleBackUPP     dbhDoubleBack;
OSType               dbhFormat;
};

```

dbhNumChannels

The number of sound channels.

dbhSampleSize

The sample size, if sound is noncompressed.

dbhCompressionID

The compression algorithm used on the samples in the compressed sound header (see below). The constant `fixedCompression` is available only with Sound Manager versions 3.0 and later. If the `compressionID` field contains the value `fixedCompression`, the Sound Manager reads the `format` field to determine the compression algorithm used to generate the compressed data. Otherwise, the Sound Manager reads the `compressionID` field. Apple reserves the right to use compression IDs in the range 0 through 511. Currently the constant `variableCompression` is not used by the Sound Manager.

dbhPacketSize

The size, in bits, of the smallest element that a given expansion algorithm can work with (see below). Beginning with Sound Manager version 3.0, you can specify the value 0 in this field to instruct the Sound Manager to determine the packet size itself.

dbhSampleRate

The sample rate.

dbhBufferPtr

Pointers to `SndDoubleBuffer` (IV–2442) structures.

dbhDoubleBack

Pointer to an `SndDoubleBackProc` (III–2148) callback.

dbhFormat

A sound compression format; see “Codec Identifiers” (IV–2655). This field is used only if the `dbhCompressionID` field contains the value `fixedCompression`.

compressionID Constants

`variableCompression`

Variable-ratio compression; value is –2.

`fixedCompression`

Fixed-ratio compression; value is –1.

SndInputCmpParam

notCompressed

Uncompressed samples; value is 0.

threeToOne

3:1 compressed samples; value is 3.

sixToOne

6:1 compressed samples; value is 4.

Discussion

The `dbhBufferPtr` array contains pointers to two structures of type `SndDoubleBuffer` (IV–2442). These are the two buffers between which the Sound Manager switches until all the sound data has been sent into the sound channel. When the call to `SndPlayDoubleBuffer` (III–1843) is made, the two buffers should both already contain a nonzero number of frames of data.

Version Notes

`SndDoubleBufferHeader` (IV–2443) is identical to `SndDoubleBufferHeader2` except that it lacks the `dbhFormat` field.

Programming Info

C interface file: `Sound.h`

See Also

See `SndDoubleBufferHeader` (IV–2443).

SndInputCmpParam

Provides data for the `SndInputReadSync` (III–1837) and `SndInputReadAsync` (III–1836) functions.

```
struct SndInputCmpParam {
    SICCompletionProcPtr    ioCompletion;
    SIInterruptProcPtr      ioInterrupt;
    OSErr                   ioResult;
    short                   pad;
    unsigned long            ioReqCount;
    unsigned long            ioActCount;
    Ptr                     ioBuffer;
    Ptr                     ioMisc;
};
```

`ioCompletion`

Pointer to a `SICCompletionProc` (III–2146) callback.

ioInterrupt
 Pointer to a SIInterruptProc (III–2147) callback.

ioResult
 I/O result error code.

pad
 Not used.

ioReqCount
Undocumented

ioActCount
Undocumented

ioBuffer
Undocumented

ioMisc
Undocumented

Associated Functions

SICCompletionProc (III–2146)

Programming Info

C interface file: Sound.h

SndListResource

Holds a sound **resource** containing sampled sound.

```
struct SndListResource {
    short      format;
    short      numModifiers;
    ModRef     modifierPart[1];
    short      numCommands;
    SndCommand commandPart[1];
    UInt8      dataPart[1];
};
```

format
Undocumented

numModifiers
Undocumented

modifierPart
Undocumented

SoundComponentData

numCommands
Undocumented
commandPart
Undocumented
dataPart
Undocumented

Programming Info

C interface file: Sound.h

SoundComponentData

Contains information about the sound format in a compressed sound file.

```
struct SoundComponentData {  
    long          flags;  
    OSType        format;  
    short         numChannels;  
    short         sampleSize;  
    UnsignedFixed sampleRate;  
    long          sampleCount;  
    Byte *        buffer;  
    long          reserved;  
};
```

flags

Normally set to 0.

format

The sound format; see “Sound Formats” (IV–2692).

numChannels

The number of channels; 1 = mono, 2 = stereo.

sampleSize

The sample size (i.e., 8 = 8-bit, 16 = 16-bit).

sampleRate

The sampling rate in samples per second.

sampleCount

The number of audio samples in the file.

buffer

Set to 0.

reserved
Set to 0.

Associated Functions

OpenMixerSoundComponent (II–1132)

Programming Info

C interface file: Sound.h

SoundComponentLink

Describes a sound component.

```
struct SoundComponentLink {
    ComponentDescription    description;
    SoundSource             mixerID;
    SoundSource *           linkID;
};
```

description

Describes the sound component.

mixerID

Reserved; do not use.

linkID

Reserved; do not use.

Programming Info

C interface file: Sound.h

SoundDescription

Describes the format of a collection of audio samples.

```
struct SoundDescription {
    long        descSize;
    long        dataFormat;
    long        resvd1;
    short       resvd2;
    short       dataRefIndex;
    short       version;
    short       revlevel;
    long        vendor;
    short       numChannels;
```

SoundDescription

```
    short        sampleSize;  
    short        compressionID;  
    short        packetSize;  
    UnsignedFixed sampleRate;  
};
```

descSize

Defines the total size, in bytes, of this structure.

dataFormat

Describes the format of the sound data; see “Sound Formats” (IV–2692). Some older movie files sometimes have a 0 value in this field. You should assume that this is the same as the 'raw ' value.

resvd1

Reserved; set to 0.

resvd2

Reserved; set to 0.

dataRefIndex

Reserved; set to 0.

version

Reserved; set to 0.

revLevel

Reserved; set to 0.

vendor

Reserved; set to 0.

numChannels

Indicates the number of sound channels used by the sound sample. Set this field to 1 for monaural sounds; set it to 2 for stereo sounds.

sampleSize

Specifies the number of bits in each sound sample. Set this field to 8 for 8-bit sound; set it to 16 for 16-bit sound.

compressionID

Reserved; set to 0.

packetSize

Reserved; set to 0.

sampleRate

Indicates the rate at which the sound samples were obtained. Sound media handlers use this value to influence the natural playback speed of the sound

described by this sound description structure. This field contains an unsigned fixed-point number that specifies the number of samples collected per second.

Associated Functions

AddSoundDescriptionExtension (I-40)

Programming Info

C interface file: Movies.h

See Also

For an extension to this structure, see the 'flap' (IV-2537) atom.

SoundDescriptionV1

Version 1 extension of the SoundDescription (IV-2449) structure.

```
struct SoundDescriptionV1 {
    SoundDescription  desc;
    unsigned long     samplesPerPacket;
    unsigned long     bytesPerPacket;
    unsigned long     bytesPerFrame;
    unsigned long     bytesPerSample;
};
```

desc

A SoundDescription (IV-2449) structure.

samplesPerPacket

The number of samples in each packet.

bytesPerPacket

The number of bytes in each packet.

bytesPerFrame

The number of bytes in each frame.

bytesPerSample

The number of bytes in each sample.

Discussion

This structure is a superset of the SoundDescription (IV-2449) structure. The added fields are taken directly from the CompressionInfo (IV-2222) structure currently used by the Sound Manager to describe the compression ratio of fixed ratio audio compression algorithms. The fields have been added to support compression algorithms which can be run at different compression ratios and to support more generic parsing of QuickTime sound tracks. They are described in detail in *Inside*

SoundHeader

Macintosh: Sound (listed in the **bibliography**). If these fields are not used, they are set to 0. File readers only need to check to see if `samplesPerPacket` is 0.

Programming Info

C interface file: `Movies.h`

See Also

See `SoundDescription` (IV–2449).

SoundHeader

Describes a set of sound samples.

```
struct SoundHeader {
    Ptr          samplePtr;
    unsigned long length;
    UnsignedFixed sampleRate;
    unsigned long loopStart;
    unsigned long loopEnd;
    UInt8        encode;
    UInt8        baseFrequency;
    UInt8        sampleArea[1];
};
```

`samplePtr`

Pointer to sound samples; if NIL, then samples are in `sampleArea`.

`length`

Length of sound samples in bytes.

`sampleRate`

Sample rate for this sound.

`loopStart`

Start of looping portion.

`loopEnd`

End of looping portion.

`encode`

Header encoding.

`baseFrequency`

The pitch of the original sampled sound.

`sampleArea`

Space for when samples follow directly.

Programming Info

C interface file: Sound.h

SoundInfoList

Passes information to SoundComponentGetInfo (III–1849) or SoundComponentSetInfo (III–1856).

```
struct SoundInfoList {
    short    count;
    Handle   infoHandle;
};
```

count

The number of elements in the array referenced by infoHandle. It is your component's responsibility to allocate the block of data referenced by that handle, but it is the caller's responsibility to dispose of that handle once it is finished with it.

infoHandle

A handle to an array of data elements. The type of these data elements depends on the kind of information requested, which is determined by the selector parameter passed to SoundComponentGetInfo or SoundComponentSetInfo.

Discussion

This structure consists of a count and a handle to a variable-sized array. The data type of the array elements depends on the kind of information being returned.

Programming Info

C interface file: Sound.h

See Also

See SoundComponentGetInfo (III–1849) and SoundComponentSetInfo (III–1856).

SoundMediaInfoHeader

Contains sound stereo balance information.

```
struct SoundMediaInfoHeader {
    long    flags;
    short   balance;
    short   rsrvd;
};
```

SoundParamBlock

flags

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

balance

Specifies the mix of sound between two stereo speakers, with a range of -1.0 to $+1.0$. The high 8 bits contain the integer part; the low 8 bits contain the fractional part. Negative values emphasize the left channel and positive values the right channel. This field is normally set to 0 for even balance.

rsrvd

Reserved; set to 0.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For the atom structure that contains the `SoundMediaInfoHeader` data structure, see 'smhd' (IV-2584).

SoundParamBlock

Describes the source data to be modified or sent to a sound output device.

```
struct SoundParamBlock {  
    long          recordSize;  
    SoundComponentData desc;  
    unsigned fixed rateMultiplier;  
    short         leftVolume;  
    short         rightVolume;  
    long          quality;  
    ComponentInstance filter;  
    SoundParamUPP moreRtn;  
    SoundParamUPP completionRtn;  
    long          refCon;  
    short         result;  
};
```

recordSize

The length, in bytes, of the sound parameter block.

desc

A `SoundComponentData` (IV-2448) structure that describes the format, size, and location of the sound data.

rateMultiplier

A multiplier to be applied to the playback rate of the sound. This field contains an unsigned fixed-point number. If, for example, this field has the value 2.0, the sound is played back at twice the rate specified in the `sampleRate` field of the `SoundComponentData` structure contained in the `desc` field.

leftVolume

The playback volume for the left channel. You specify a volume with a 16-bit value, where 0x0000 represents no volume and 0x0100 represents full volume. You can overdrive a channel's volume by passing volume levels greater than 0x0100.

rightVolume

The playback volume for the right channel. You specify a volume with a 16-bit value, where 0x0000 represents no volume and 0x0100 represents full volume. You can overdrive a channel's volume by passing volume levels greater than 0x0100.

quality

The level of quality for the sound. This value usually determines how much processing should be applied during audio data processing (such as rate conversion and decompression) to increase the output quality of the sound.

filter

Reserved for future use. You should set this field to `NIL`.

moreRtn

A pointer to a `SoundParamProc` (III–2149) callback that is called to retrieve another buffer of audio data. This field is used internally by the Sound Manager.

completionRtn

A pointer to a `SoundParamProc` (III–2149) callback that is called when the sound has finished playing. This field is used internally by the Sound Manager.

refCon

A value that is to be passed to the callback routines specified in the `moreRtn` and `completionRtn` fields. You can use this field to pass information (for example, the address of a structure) to a callback routine.

result

The status of the sound that is playing. The value 1 indicates that the sound is currently playing. The value 0 indicates that the sound has finished playing. Any negative value indicates that some error has occurred.

Associated Functions

`SoundComponentPlaySourceBuffer` (III–1854)

SoundSlopeAndInterceptRecord

Programming Info

C interface file: Sound.h

See Also

See SoundComponentData (IV–2448).

SoundSlopeAndInterceptRecord

Provides floating-point conversion variables for a 'flap' (IV–2537) atom.

```
struct SoundSlopeAndInterceptRecord {  
    Float64    slope;  
    Float64    intercept;  
    Float64    minClip;  
    Float64    maxClip;  
};
```

slope

Undocumented

intercept

Undocumented

minClip

Undocumented

maxClip

Undocumented

Programming Info

C interface file: Sound.h

See Also

See 'flap' (IV–2537).

SPB

Passes information to low-level sound input functions.

```
struct SPB {  
    long                inRefNum;  
    unsigned long       count;  
    unsigned long       milliseconds;  
    unsigned long       bufferLength;  
    Ptr                 bufferPtr;  
    SICompletionUPP     completionRoutine;
```

```

    SIInterruptUPP    interruptRoutine;
    long              userLong;
    OSErr             error;
    long              unused1;
};

```

inRefNum

The reference number of the sound input device from which recording is to occur.

count

The number of bytes to record. After recording finishes, the `count` field indicates the number of bytes actually recorded.

milliseconds

The number of milliseconds to record. If the `count` and `milliseconds` fields do not agree, then the field that specifies the longer recording time is used. After recording finishes, the `milliseconds` field indicates the number of milliseconds actually recorded.

bufferLength

The length in bytes of the buffer into which the recorded sound data is to be placed. If the buffer is shorter than the recording time, then the recording time is truncated so that the recorded data can fit into the buffer.

bufferPtr

A pointer to the buffer to store sound data in.

completionRoutine

Pointer to an `SICompletionProc` (III–2146) callback.

interruptRoutine

Pointer to an `SIInterruptProc` (III–2147) callback.

userLong

A long integer for your application's use. You can use this field, for instance, to pass a handle to an application-defined structure to the sound input completion or interrupt routine.

error

Contains a value greater than 0 while recording unless an error occurs, in which case it contains a value less than 0. Your application can poll this field to check on the status of an asynchronous recording. If recording terminates without an error, this field contains 0. See “Error Codes” (IV–2663).

unused1

Reserved; set to 0.

SpriteDescription

Associated Functions

SICompletionProc (III–2146)

Programming Info

C interface file: Sound.h

See Also

For a function that uses this structure, see SPBRecord (III–1878).

SpriteDescription

Provides information to compress a **sprite** using a data compression codec.

```
struct SpriteDescription {  
    long    descSize;  
    long    dataFormat;  
    long    resvd1;  
    short   resvd2;  
    short   dataRefIndex;  
    long    version;  
    OSType  decompressorType;  
    long    sampleFlags;  
};
```

descSize

The total size of the SpriteDescription structure, including extra data.

dataFormat

Undocumented

resvd1

Reserved; do not use.

resvd2

Reserved; do not use.

dataRefIndex

Undocumented

version

Undocumented

decompressorType

A data decompressor component type, or 0 for no decompression; see “Codec Identifiers” (IV–2655). If this field is nonzero, a component of the specified type is used to decompress the **sprite** sample when it is loaded.

sampleFlags

Undocumented

Programming Info

C interface file: Movies.h

SpriteRecord

Contains a **sprite**.

```
struct SpriteRecord {
    long    data[1];
};
```

data

An array of **sprite** data.

Programming Info

C interface file: Movies.h

SpriteWorldRecord

Contains a **sprite** world.

```
struct SpriteWorldRecord {
    long    data[1];
};
```

data

An array of **sprite** world data.

Programming Info

C interface file: Movies.h

SProcRec

A link in a list that begins with the gdSearchProc field of the GDevice (IV–2264) structure.

```
struct SProcRec {
    Handle          nxtSrch;
    ColorSearchUPP srchProc;
```

SSpLocalizationData

```
};
```

nxtSrch

A handle to next SProcRec structure.

srchProc

Pointer to a custom search procedure; see *Advanced Color Imaging on the Mac OS* (listed in the **bibliography**), Chapter 7.

Programming Info

C interface file: Quickdraw.h

SSpLocalizationData

Provides a format for sound localization data.

```
struct SSpLocalizationData {
    UInt32          cpuLoad;
    UInt32          medium;
    float           humidity;
    float           roomSize;
    float           roomReflectivity;
    float           reverbAttenuation;
    UInt32          sourceMode;
    float           referenceDistance;
    float           coneAngleCos;
    float           coneAttenuation;
    SSpLocationData currentLocation;
    UInt32          reserved0;
    UInt32          reserved1;
    UInt32          reserved2;
    UInt32          reserved3;
    UInt32          virtualSourceCount;
    SSpVirtualSourceData virtualSource[4];
};
```

cpuLoad

CPU load versus quality; 0 is best.

medium

Medium for sound propagation.

humidity

Humidity when medium is air.

roomSize

Reverberation model: distance between walls.

roomReflectivity
Reverberation model: bounce attenuation.

reverbAttenuation
Reverberation model: mix level.

sourceMode
Type of filtering to apply.

referenceDistance
Nominal distance for recording.

coneAngleCos
Cos (angle/2) of attenuation cone.

coneAttenuation
Attenuation outside the cone.

currentLocation
Location of the sound.

reserved0
Reserved; set to 0.

reserved1
Reserved; set to 0.

reserved2
Reserved; set to 0.

reserved3
Reserved; set to 0.

virtualSourceCount
Number of reflections.

virtualSource
The reflections.

Programming Info

C interface file: SoundSprocket.h

StandardFileReply

Returns the results of user actions in the Standard File dialog box.

```
struct StandardFileReply {
    Boolean      sfGood;
    Boolean      sfReplacing;
```

StandardFileReply

```
OSType      sfType;  
FSSpec      sfFile;  
ScriptCode  sfScript;  
short       sfFlags;  
Boolean     sfIsFolder;  
Boolean     sfIsVolume;  
long        sfReserved1;  
short       sfReserved2;  
};
```

sfGood

Whether or not the reply record is valid; that is, whether your application can use the information in the other fields. The field is set to `TRUE` after the user clicks Save or Open, and to `FALSE` after the user clicks Cancel.

sfReplacing

Whether or not a file to be saved replaces an existing file of the same name. Your application can use the value of this field instead of checking for and handling name conflicts itself.

sfType

The file type; see “File Types and Creators” (IV–2668).

sfFile

The file or folder selected by the user. If the selected file is a stationery pad, the `FSSpec` describes the file itself, not a copy of the file.

sfScript

The file’s script code; see “Localization Codes” (IV–2673).

sfFlags

Mac OS Finder flags for the selected file or folder.

sfIsFolder

`TRUE` if the selected item is a folder.

sfIsVolume

`TRUE` if the selected item is a volume.

sfReserved1

Reserved; do not use.

sfReserved2

Reserved; do not use.

Discussion

The **Standard File Package** fills in the reply record and returns when the user completes one of its dialog boxes, either by selecting a file and clicking Save or

Open, or by clicking Cancel. Your application checks the values in the reply record to see what action to take, if any.

Associated Functions

CustomGetFilePreview (I–163)

Programming Info

C interface file: `StandardFile.h`

See Also

See *Inside Macintosh: Files* (listed in the **bibliography**).

StateBlock

Provides information for the `stateVars` field of a `CmpSoundHeader` (IV–2176) structure.

```
struct StateBlock {
    short    stateVar[64];
};
```

`stateVar`

An array of integers. This field contains state variables that need to be preserved across invocations of the compression algorithm. The size of this field is defined by a constant (see below).

`stateVar` Size Constant

`stateBlockSize`

Value is 64.

Associated Functions

`Comp3to1` (I–104)

Programming Info

C interface file: `Sound.h`

StringRangeRecord

Undocumented

```
struct StringRangeRecord {
    long    maxChars;
    long    maxLines;
};
```

SynthesizerConnections

maxChars

The maximum length of the string.

maxLines

The number of editing lines to use (1 typical, 0 to default).

Programming Info

C interface file: ImageCodec.h

SynthesizerConnections

Describes how a MIDI device is connected to the user's computer.

```
struct SynthesizerConnections {  
    OSType    clientID;  
    OSType    inputPortID;  
    OSType    outputPortID;  
    long      midiChannel;  
    long      flags;  
    long      unique;  
    long      reserved1;  
    long      reserved2;  
};
```

clientID

The client ID provided by the MIDI Manager, or 'OMS ' for an OMS port.

inputPortID

The ID provided by the MIDI Manager or OMS for the port used to SEND to the MIDI synthesizer.

outputPortID

The ID provided by the MIDI Manager or OMS for the port that RECEIVES from a keyboard or other control device.

midiChannel

The system MIDI channel or, for a hardware device, the slot number.

flags

Constants (see below) that provide information about the type of connection.

unique

A unique ID you can use instead of an index to identify the synthesizer to the note allocator.

reserved1

Reserved. Set to 0.

reserved2

Reserved. Set to 0.

flags Constants

kSynthesizerConnectionMono

If set, and the synthesizer can be both monophonic and polyphonic, the synthesizer is instructed to take up its channels sequentially from the system channel in monophonic mode.

kSynthesizerConnectionMMgr

This connection is imported from the MIDI Manager.

kSynthesizerConnectionOMS

This connection is imported from the Open Music System (OMS).

kSynthesizerConnectionQT

This connection is a QuickTime-only port.

kSynthesizerConnectionFMS

This connection is imported from the FreeMIDI system.

Associated Functions

NAGetDefaultMIDIInput (II-1023)

Programming Info

C interface file: QuickTimeMusic.h

SynthesizerDescription

Contains information about a synthesizer.

```
struct SynthesizerDescription {
    OSType          synthesizerType;
    Str31           name;
    unsigned long   flags;
    unsigned long   voiceCount;
    unsigned long   partCount;
    unsigned long   instrumentCount;
    unsigned long   modifiableInstrumentCount;
    unsigned long   channelMask;
    unsigned long   drumPartCount;
    unsigned long   drumCount;
    unsigned long   modifiableDrumCount;
    unsigned long   drumChannelMask;
    unsigned long   outputCount;
    unsigned long   latency;
    unsigned long   controllers[4];
    unsigned long   gmInstruments[4];
}
```

SynthesizerDescription

```
    unsigned long    gmDrums[4];  
};
```

synthesizerType

The synthesizer type. This is the same as the music component subtype.

name

Text name of the synthesizer type.

flags

Constants (see below) that provide information about how the synthesizer works.

voiceCount

Maximum polyphony.

partCount

Maximum multi-timbrality (and MIDI channels).

instrumentCount

The number of built-in ROM instruments. This does not include General MIDI instruments.

modifiableInstrumentCount

The number of slots available for saving user-modified instruments.

channelMask

Which channels a MIDI device always uses for instruments. Set to 0xFFFF for all channels.

drumPartCount

The maximum multi-timbrality of drum parts. For synthesizers where drum kits are separated from instruments.

drumCount

The number of built-in ROM drum kits. This does not include General MIDI drum kits. For synthesizers where drum kits are separated from instruments.

modifiableDrumCount

The number of slots available for saving user-modified drum kits. For MIDI synthesizers where drum kits are separated from instruments.

drumChannelMask

Which channels a MIDI device always uses for drum kits. Set to FFFF for all channels.

outputCount

The number of audio outputs. This is usually 2.

latency

The response time in microseconds.

controllers

An array of 128 bits identifying the available controllers; see “Music Controllers” (IV–2682). Bits are numbered from 1 to 128, starting with the most significant bit of the long word and continuing to the least significant of the last bit.

gmInstruments

An array of 128 bits giving the available General MIDI instruments.

gmDrums

An array of 128 bits giving the available General MIDI drum kits.

flags Constants

kSynthesizerDynamicVoice

Voices can be assigned to parts on the fly with this synthesizer (otherwise, polyphony is very important).

kSynthesizerUsesMIDIPort

This synthesizer must be patched through a MIDI system, such as the MIDI Manager or OMS.

kSynthesizerMicrotone

This synthesizer can play microtonal scales.

kSynthesizerHasSamples

This synthesizer has some use for sampled audio data.

kSynthesizerMixedDrums

Any part of this synthesizer can play drum parts.

kSynthesizerSoftware

This synthesizer is implemented in main CPU software and uses CPU cycles.

kSynthesizerHardware

This synthesizer is a hardware device, not a software synthesizer or MIDI device.

kSynthesizerDynamicChannel

This synthesizer can move any part to any channel or disable each part. For devices only.

kSynthesizerHogsSystemChannel

Even if the kSynthesizerDynamicChannel bit is set, this synthesizer always responds on its system channel. For MIDI devices only.

TCTextOptions

kSynthesizerSlowSetPart

This synthesizer does not respond rapidly to the various set part and set part instrument calls.

kSynthesizerOffline

This synthesizer can enter an offline synthesis mode.

kSynthesizerGM

This synthesizer is a General MIDI device.

Associated Functions

MusicGetDescription (II-986)

Programming Info

C interface file: QuickTimeMusic.h

TCTextOptions

Holds text font and style information.

```
struct TCTextOptions {
    short    txFont;
    short    txFace;
    short    txSize;
    short    pad;
    RGBColor foreColor;
    RGBColor backColor;
};
```

txFont

Specifies the number of the font.

txFace

Specifies the font's style (bold, italic, and so on).

txSize

Specifies the font's size.

pad

Unused field to make structure long-word aligned.

foreColor

Specifies the foreground color.

backColor

Specifies the background color.

Associated Functions

TGetDisplayOptions (III–1918)

Programming Info

C interface file: QuickTimeComponents.h

TERec

Contains display, storage, styling, and other information for text editing.

```

struct TERec {
    Rect          destRect;
    Rect          viewRect;
    Rect          selRect;
    short         lineHeight;
    short         fontAscent;
    Point         selPoint;
    short         selStart;
    short         selEnd;
    short         active;
    WordBreakUPP wordBreak;
    TClickLoopUPP clickLoop;
    long          clickTime;
    short         clickLoc;
    long          caretTime;
    short         caretState;
    short         just;
    short         teLength;
    Handle        hText;
    long          hDispatchRec;
    short         cliKStuff;
    short         crOnly;
    short         txFont;
    StyleField    txFace;
    short         txMode;
    short         txSize;
    GrafPtr       inPort;
    HighHookUPP   highHook;
    CaretHookUPP  caretHook;
    short         nLines;
    short         lineStarts[16001];
};

```

destRect

The destination rectangle, in local coordinates.

`viewRect`

The view rectangle, in local coordinates.

`selRect`

The selection rectangle, whose boundaries are defined in local coordinates. This value is the current selection range or insertion point.

`lineHeight`

The vertical spacing of lines of text. Vertical spacing may be fixed or it may vary from line to line, depending upon specific text attributes. If the value of `lineHeight` is greater than 0, this field specifies the fixed vertical distance from the ascent line of one line of text down to the ascent line of the next. If the value of `lineHeight` is less than 1, then this field specifies the vertical distance from the ascent line of one line of text down to the ascent line of the next calculated independently for each line, based on the maximum value for any individual character attribute on that line.

`fontAscent`

The font ascent line. If the value of `fontAscent` is greater than 0, this field specifies how far above the base line the pen is positioned to begin drawing the caret or highlighting. For single-spaced text, this is the height of the text in pixels (the height of the tallest characters in the font from the base line). If the value of `fontAscent` is less than 1, this field specifies the font ascent calculated independently for each line, based on maximum value for any individual character attribute on that line.

`selPoint`

The point selected with the mouse, in the local coordinates of the current graphics port. The assembly-language offset for this field is named `teSelPoint`.

`selStart`

The byte offset of the beginning of a selection range. Note that byte offset 0 refers to the first byte in the text buffer.

`selEnd`

The byte offset of the end of a selection range. To include that byte, this value must be 1 greater than the position of the last byte offset of the text.

`active`

Used internally.

`wordBreak`

The record's word selection break routine. This routine determines the word that is highlighted when the user double-clicks in the text and the position at which text is wrapped at the end of a line.

`clickLoop`

The pointer to the click loop routine. The specified click loop routine is called repeatedly as long as the mouse button is held down within the text.

`clickTime`

Used internally.

`clickLoc`

Used internally.

`caretTime`

Used internally.

`caretState`

Used internally.

`just`

The type of text alignment: default (according to the primary line direction), left, center, or right.

`teLength`

The number of bytes in the text to be edited. For two-byte systems, potentially twice the number of characters. Initially set to zero. The maximum length is 32767 bytes.

`hText`

A handle to the text. Initially, it points to a zero-length block of text in the heap.

`hDispatchRec`

Used internally.

`clickStuff`

Used internally.

`crOnly`

A value specifying whether or not text wraps at the right edge of the destination rectangle. If `crOnly` is positive, text does wrap. If `crOnly` is negative, new lines are specified explicitly by Return characters only; text does not wrap at the edge of the destination rectangle. (This is useful in an application similar to a programming-language editor, where you may not want a single line of code to be split onto two lines.)

`txFont`

The font of all the text in the `TERec` structure if the `txSize` field of this `TERec` structure is greater than or equal to 0. If you change this value, the entire text of this `TERec` structure has the new characteristic when it is redrawn; also, remember to change the `lineHeight` and `fontAscent` fields as well. If the `txSize`

TERec

field is `-1`, this field combines with `txFace` to hold a handle to the associated `TEStyleRec` (IV-2473) structure.

`txFace`

The character attributes of all the text in an `TERec` structure if the `txSize` field of this `TERec` structure is greater than or equal to 0. If you change this value, the entire text of this `TERec` structure has the new characteristic when it is redrawn; also, remember to change the `lineHeight` and `fontAscent` fields as well. If the `txSize` field is `-1`, this field combines with `txFont` to hold a handle to the associated `TEStyleRec` (IV-2473) structure.

`txMode`

The pen mode of all the text in the `TERec` structure. If you change this value, the entire text of this `TERec` structure has the new characteristic when it is redrawn; also, remember to change the `lineHeight` and `fontAscent` fields as well.

`txSize`

Depending on its value, `txSize` either contains the point size of all of the text or it acts as a flag indicating whether or not there is associated character attribute information. If `txSize` is greater than or equal to 0, this is a monostyled `TERec` structure, that is, all text is set in a single font, size, and face, and the value of `txSize` is the size of the text. If `txSize` is `-1`, the `TERec` structure contains associated character attribute information and the `txFont` and `txFace` fields combine to form a handle to the `TEStyleRec` (IV-2473) structure.

`inPort`

A pointer to the `CGrafPort` (IV-2168) structure associated with this `TERec` structure.

`highHook`

A pointer to the routine that deals with text highlighting.

`caretHook`

A pointer to the routine that controls the appearance of the caret.

`nLines`

The number of lines in the text.

`lineStarts`

An array containing the character position of the first character in each line. This is a dynamic data structure having only as many elements as needed. `TextEdit` calculates these values internally, so do not change the elements of the `lineStarts` array. Because this data structure grows and shrinks, the size of the `TERec` structure changes.

Programming Info

C interface file: TextEdit.h

TEStyleRec

Stores the character attribute information for the text of a multistyled TERec (IV–2469) structure.

```
struct TEStyleRec {
    short      nRuns;
    short      nStyles;
    STHandle    styleTab;
    LHHandle    lhTab;
    long        teRefCon;
    NullStHandle nullStyle;
    StyleRun    runs[8001];
};
```

nRuns

The number of style runs in the text.

nStyles

The number of distinct sets of character attributes used in the text; this forms the size of the style table.

styleTab

A handle to the style table.

lhTab

A handle to the line height table.

teRefCon

A reference constant for use by applications. The application can use this 32-bit field to suit its needs.

nullStyle

A handle to the style scrap record used to store the character attribute information for a null selection.

runs

A table of style runs that is of indefinite length. To determine the length of a run, you subtract its start position from that of the next entry in the style run table. A dummy entry at the end of the style run table delimits the length of the last run; its start position is equal to the overall number of characters in the text, plus 1.

TextDescription

Discussion

If an edit record has associated character attribute information, its `txFont` and `txFace` fields combine to hold a style handle to its style record. The text is divided into style runs, summarized in the style run table which is part of the style record. Each entry in the style run table gives the starting character position of a run and an index into the style table. The style table element pointed to by the style run index describes the character attributes for that run.

Programming Info

C interface file: `TextEdit.h`

TextDescription

Describes a sample of text.

```
struct TextDescription {
    long          descSize;
    long          dataFormat;
    long          resvd1;
    short         resvd2;
    short         dataRefIndex;
    long          displayFlags;
    long          textJustification;
    RGBColor      bgColor;
    Rect          defaultTextBox;
    ScrpSTElement defaultStyle;
    char          defaultFontName[1];
};
```

`size`

Defines the total size of this text description structure.

`type`

Indicates the type (data type 'text').

`resvd1`

Reserved; set to 0.

`resvd2`

Reserved; set to 0.

`displayFlags`

Contains flags (see below) that specify how the text is to be displayed.

`textJustification`

Contains a constant (see below) that specifies how the text is to be aligned.

bgColor

Specifies the background color for the text display.

defaultTextBox

Indicates the location of the text within track boundaries.

defaultStyle

Provides a `ScrpSTElement` (IV–2422) structure that specifies the default style for the text display.

defaultFontName

The name of the default font.

displayFlags Constants

dfDontDisplay

Does not display the specified sample.

dfDontAutoScale

Does not scale the text if the track bounds increase.

dfClipToTextBox

Clips to just the text box. This is useful if the text overlays the video.

dfShrinkTextBoxToFit

Recalculates size of the `textBox` parameter to just fit the given text and stores this rectangle with the text data.

dfScrollIn

Scrolls the text in until the last of the text is in view. This flag is associated with the `scrollDelay` parameter.

dfScrollOut

Scrolls text out until the last of the text is out of view. This flag is associated with the `scrollDelay` parameter. If both `dfScrollIn` and `dfScrollOut` are set, the text is scrolled in, then out.

dfHorizScroll

Scrolls a single line of text horizontally. If the `dfHorizScroll` flag is not set, then the scrolling is vertical.

dfReverseScroll

If set, scrolls vertically down, rather than up. If not set, horizontal scrolling proceeds toward the left rather than toward the right.

textJustification Constants

teFlushDefault

Default alignment according to the primary line direction.

teCenter

Center for all scripts.

TextDisplayData

teFlushRight

Right alignment for all scripts.

teFlushLeft

Left alignment for all scripts.

Associated Functions

TextMediaDrawRaw (III–1940)

Programming Info

C interface file: Movies.h

TextDisplayData

Contains formatting information for a text sample.

```
struct TextDisplayData {
    long          displayFlags;
    long          textJustification;
    RGBColor      bgColor;
    Rect          textBox;
    short         beginHilite;
    short         endHilite;
    RGBColor      hiliteColor;
    Boolean       doHiliteColor;
    SInt8         filler;
    TimeValue     scrollDelayDur;
    Point         dropShadowOffset;
    short         dropShadowTransparency;
};
```

displayFlags

Contains flags (see below) that represent the values of text descriptors.

textJustification

Contains constants (see below) that specify the alignment of the text in the text box. Possible values are teFlushDefault, teCenter, teFlushRight, and teFlushLeft. For more information on text alignment and the text justification constants, see the “TextEdit” chapter of *Inside Macintosh: Text* (listed in the **bibliography**).

bgColor

Specifies the background color of the rectangle specified by the textBox field. The background color is specified as an RGB color value.

`textBox`

Specifies the rectangle of the text box.

`beginHilite`

Specifies the one-based index of the first character in the sample to highlight.

`endHilite`

Specifies the one-based index of the last character in the sample to highlight.

`doHiliteColor`

Specifies whether to use the color specified by the `hiliteColor` field for highlighting. If the value of this field is `TRUE`, the highlight color is used for highlighting. If the value of this field is `FALSE`, reverse video is used for highlighting.

`filler`

Reserved.

`scrollDelayDur`

Specifies a scroll delay. The scroll delay is specified as the number of units of delay in the text track's time scale. For example, if the time scale is 600, a scroll delay of 600 causes the sample text to be delayed one second. In order for this field to take effect, scrolling must be enabled.

`dropShadowOffset`

Specifies an offset for the drop shadow. For example, if the point specified is (3,4), the drop shadow is offset 3 pixels to the right and 4 pixels down. In order for this field to take effect, drop shadowing must be enabled.

`dropShadowTransparency`

Specifies the intensity of the drop shadow as a value between 0 and 255. In order for this field to take effect, drop shadowing must be enabled.

displayFlags Constants

`dfDontDisplay`

Does not display the specified sample.

`dfDontAutoScale`

Does not scale the text if the track bounds increase.

`dfClipToTextBox`

Clips to just the text box. This is useful if the text overlays the video.

`dfShrinkTextBoxToFit`

Recalculates size of the `textBox` parameter to just fit the given text and stores this rectangle with the text data.

ThreeDeeDescription

`dfScrollIn`

Scrolls the text in until the last of the text is in view. This flag is associated with the `scrollDelay` parameter.

`dfScrollOut`

Scrolls text out until the last of the text is out of view. This flag is associated with the `scrollDelay` parameter. If both `dfScrollIn` and `dfScrollOut` are set, the text is scrolled in, then out.

`dfHorizScroll`

Scrolls a single line of text horizontally. If the `dfHorizScroll` flag is not set, then the scrolling is vertical.

`dfReverseScroll`

If set, scrolls vertically down, rather than up. If not set, horizontal scrolling proceeds toward the left rather than toward the right.

textJustification Constants

`teFlushDefault`

Default alignment according to the primary line direction.

`teCenter`

Center for all scripts.

`teFlushRight`

Right alignment for all scripts.

`teFlushLeft`

Left alignment for all scripts.

Discussion

When the text export component exports a text sample, it uses the information in this structure to generate the appropriate text descriptors for the sample. Likewise, when the text import component imports a text sample, it sets the appropriate fields in this structure based on the sample's text descriptors.

Associated Functions

`TextExportGetDisplayData` (III–1928)

Programming Info

C interface file: `QuickTimeComponents.h`

ThreeDeeDescription

Undocumented

```
struct ThreeDeeDescription {
```

```

    long    descSize;
    long    dataFormat;
    long    resvd1;
    short   resvd2;
    short   dataRefIndex;
    long    version;
    long    rendererType;
    long    decompressorType;
};

```

descSize

The total size of this structure, including extra data.

dataFormat

Undocumented

resvd1

Reserved; do not use.

resvd2

Reserved; do not use.

dataRefIndex

Undocumented

version

Which version is this data.

rendererType

Which renderer to use; 0 for default.

decompressorType

Which decompressor to use; 0 for default.

Programming Info

C interface file: `Movies.h`

ThreeDeeNonLinearSample

Undocumented

```

struct ThreeDeeNonLinearSample {
    float      DurFromLastSample;
    TQ3Matrix4x4 matrix;
};

```

DurFromLastSample

Duration from 0 to 1.

ThreeDeeNonLinearTweenHeaderAtom

matrix

Undocumented

Programming Info

C interface file: Movies.h

ThreeDeeNonLinearTweenHeaderAtom

Undocumented

```
struct ThreeDeeNonLinearTweenHeaderAtom {  
    long    number;  
    long    dataSize;  
    float    tensionFactor;  
    long    reserved1;  
    long    reserved2;  
};
```

number

Undocumented

dataSize

Undocumented

tensionFactor

Undocumented; default is 0.

reserved1

Undocumented

reserved2

Undocumented

Programming Info

C interface file: Movies.h

ThreeDeeVRObjectSample

Undocumented

```
struct ThreeDeeVRObjectSample {  
    long            rows;  
    long            columns;  
    TQ3Matrix4x4    calib1;  
    TQ3Matrix4x4    calib2;
```

```
        long          reserved1;
        long          reserved2;
};
```

rows
Undocumented

columns
Undocumented

calib1
Undocumented

calib2
Undocumented

reserved1
Undocumented

reserved2
Undocumented

Programming Info
C interface file: Movies.h

TimeBaseRecord

Contains a time base.

```
struct TimeBaseRecord {
    long    data[1];
};
```

data
Array of data that constitutes a time base.

Programming Info
C interface file: Movies.h

TimeCodeCounter

Provides frame count information for a TimeCodeRecord (IV-2484) structure.

```
struct TimeCodeCounter {
    long    counter;
```

TimeCodeDef

```
};
```

counter

A frame count.

Programming Info

C interface file: QuickTimeComponents.h

See Also

See TimeCodeRecord (IV–2484).

TimeCodeDef

Contains timecode format information.

```
struct TimeCodeDef {
    long          flags;
    TimeScale     fTimeScale;
    TimeValue     frameDuration;
    UInt8         numFrames;
    UInt8         padding;
};
```

flags

Contains flags (see below) that provide timecode format information.

fTimeScale

Contains the time scale for interpreting the `frameDuration` field. This field indicates the number of time units per second.

frameDuration

Specifies how long each frame lasts, in the units defined by the `fTimeScale` field.

numFrames

Indicates the number of frames stored per second. In the case of timecodes that are interpreted as counters, this field indicates the number of frames stored per timer “tick.”

padding

Unused.

flags Constants

tcDropFrame

Indicates that the timecode drops frames occasionally to stay in synchronization. Some timecodes run at other than a whole number of frames per second. For example, NTSC video runs at 29.97 frames per second. In order

to resynchronize between the timecode rate and a 30 frames-per-second playback rate, the timecode drops a frame at a predictable time (in much the same way that leap years keep the calendar synchronized).

`tc24HourMax`

Indicates that the timecode values return to 0 at 24 hours.

`tcNegTimesOK`

Indicates that the timecode supports negative time values.

`tcCounter`

Indicates that the timecode should be interpreted as a simple counter, rather than as a time value. This allows the timecode to contain either time information or counter (such as a tape counter) information.

Associated Functions

`TCFrameNumberToTimeCode` (III–1916)

Programming Info

C interface file: `QuickTimeComponents.h`

TimeCodeDescription

Defines the format and content of a timecode media sample description.

```
struct TimeCodeDescription {
    long        descSize;
    long        dataFormat;
    long        resvd1;
    short       resvd2;
    short       dataRefIndex;
    long        flags;
    TimeCodeDef timeCodeDef;
    long        srcRef[1];
};
```

`descSize`

Specifies the size of the sample description, in bytes.

`dataFormat`

Indicates the sample description type; see “Component Identifiers” (IV–2657).
`TimeCodeMediaType = 'tmcd'.`

`resvd1`

Reserved; set to 0.

TimeCodeRecord

resvd2

Reserved; set to 0.

dataRefIndex

Contains an index value indicating which of the media's data references contains the sample data for this sample description.

flags

Reserved; set to 0.

timeCodeDef

Contains a timecode definition structure that defines timecode format information.

srcRef

Contains the timecode's source information. This is formatted as a user data item that is stored in the sample description. The media handler provides functions that allow you to get and set this data.

Associated Functions

TCGetSourceRef (III–1919)

Programming Info

C interface file: QuickTimeComponents.h

TimeCodeRecord

Interprets time information as both a time value (HH:MM:SS:FF) and a frame count.

```
union TimeCodeRecord {  
    TimeCodeTime      t;  
    TimeCodeCounter    c;  
};
```

t

The timecode value interpreted as time in a TimeCodeTime (IV–2485) structure.

c

The timecode value interpreted as a frame count in a TimeCodeCounter (IV–2481) structure.

Discussion

When you use the timecode media handler to work with time values, the media handler uses TimeCodeRecord structures to store the time values. These structures allows you to interpret the time information as either a time value (HH:MM:SS:FF) or a counter value. Given a timecode definition, you can freely convert from frame

numbers to time values and from time values to frame numbers. For a time value of 00:00:12:15 (HH:MM:SS:FF), you would obtain a frame number of 375 ($(12*30) + 15$).

Associated Functions

TCFrameNumberToTimeCode (III–1916)

Programming Info

C interface file: QuickTimeComponents.h

See Also

See TimeCodeTime (IV–2485) and TimeCodeCounter (IV–2481).

TimeCodeTime

Provides time information for a TimeCodeRecord (IV–2484) structure.

```
struct TimeCodeTime {
    UInt8    hours;
    UInt8    minutes;
    UInt8    seconds;
    UInt8    frames;
};
```

hours

Hours in the HH:MM:SS:FF time format.

minutes

Minutes in the HH:MM:SS:FF time format. With timecodes that allow negative time values, this field (addressed `TimeCodeRecord.t.minutes`) indicates whether the time value is positive or negative. If the `tctNegFlag` bit (see below) of the minutes field is set to 1, the time value is negative.

seconds

Seconds in the HH:MM:SS:FF time format.

frames

Frames in the HH:MM:SS:FF time format.

minutes Flag Constant

`tctNegFlag`

If this bit of the minutes field is set to 1, the time value in `TimeCodeTime` is negative.

Programming Info

C interface file: QuickTimeComponents.h

TimeRecord

See Also

See [TimeCodeRecord \(IV–2484\)](#).

TimeRecord

Contains a time value with its scale and time base.

```
struct TimeRecord {
    CompTimeValue    value;
    TimeScale        scale;
    TimeBase         base;
};
```

value

Contains the time value. The time value defines either a duration or an absolute time by specifying the corresponding number of units of time. For durations, this is the number of time units in the period. For an absolute time, this is the number of time units since the beginning of the time coordinate system. The unit for this value is defined by the `scale` field. The time value is expressed as a 64-bit integer quantity. This 64-bit quantity consists of two 32-bit integers and is defined by the `Int64` data type.

scale

Contains the time scale. This field specifies the number of units of time that pass each second. If you specify a value of 0, the time base uses its natural time scale.

base

Contains a reference to the time base. You obtain a time base by calling [GetMovieTimeBase \(I–496\)](#) or [NewTimeBase \(II–1119\)](#). If the **time structure** defines a duration, set this field to `NIL`. Otherwise, this field must refer to a valid time base.

Associated Functions

[AddTime \(I–41\)](#)

Programming Info

C interface file: `Movies.h`

TimeToSampleNum

Maps a sample number to its sample duration.

```
struct TimeToSampleNum {
```

```

    long        sampleCount;
    TimeValue    sampleDuration;
};

```

sampleCount

A physical sample number.

sampleDuration

The physical duration of the sample.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For an atom that contains `TimeToSampleNum` structures, see 'stts' (IV-2595).

ToneDescription

Provides the information needed to produce a specific musical sound.

```

struct ToneDescription {
    BigEndianOSType    synthesizerType;
    Str31              synthesizerName;
    Str31              instrumentName;
    BigEndianLong      instrumentNumber;
    BigEndianLong      gmNumber;
};

```

synthesizerType

A synthesizer type constant (see below). A value of 0 specifies that any type of synthesizer is acceptable.

synthesizerName

The name of the instrument to use.

instrumentName

The name of the instrument to use.

instrumentNumber

The instrument number of the instrument to use. This value, which must be in the range 1–262143, can specify General MIDI and GS instruments as well as other instruments. The instrument specified by this field is used if it is available; if not, the instrument specified by the `gmNumber` field is used. If neither of the instruments specified by the `instrumentNumber` or `gmNumber` fields is available, the instrument specified by the `instrumentName` field is used. Finally, if none of these fields specifies an instrument that is available, no tone is played.

TQ3Matrix4x4

gmNumber

The instrument number of a General MIDI or GS instrument to use if the instrument specified by the `instrumentNumber` field is not available. This value, which must be in the range 1–16383, can specify only General MIDI and GS instruments. The instrument specified by the `instrumentNumber` field is used if it is available; if not, the instrument specified by the `gmNumber` field is used. If neither of the instruments specified by the `instrumentNumber` or `gmNumber` fields is available, the instrument specified by the `instrumentName` field is used. Finally, if none of these fields specifies an instrument that is available, no tone is played.

synthesizerType Constants

kSoftSynthComponentSubType

Software synthesizer; value is 'ss'.

kGMSynthComponentSubType

General MIDI synthesizer; value is 'gm'.

Discussion

The tune header in the QuickTime Music Architecture has a `ToneDescription` structure for each instrument used. These structures are also used in the tone description atoms of atomic instruments.

Associated Functions

`MusicFindTone` (II–981)

Programming Info

C interface file: `QuickTimeMusic.h`

TQ3Matrix4x4

Defines a 4-by-4 matrix transformation.

```
struct TQ3Matrix4x4 {  
    float    value[4][4];  
};
```

value

The matrix.

Programming Info

C interface file: `QD3D.h`

TrackDirectoryEntry

Includes a track in a movie.

```
struct TrackDirectoryEntry {
    TrackDirectory    trackDirectory;
};
```

trackDirectory

A 'trak' (IV-2600) atom.

Programming Info

C interface file: `MoviesFormat.h`

See Also

See the 'trak' (IV-2600) and 'moov' (IV-2566) atoms.

TrackEditStateRecord

Contains a track edit state.

```
struct TrackEditStateRecord {
    long    data[1];
};
```

data

An array of data that constitutes a track edit state.

Programming Info

C interface file: `Movies.h`

TrackHeader

Specifies the characteristics of a track in a movie.

```
struct TrackHeader {
    long    flags;
    long    creationTime;
    long    modificationTime;
    long    trackID;
    long    reserved1;
    TimeValue    duration;
    long    reserved2;
    long    reserved3;
```

TrackHeader

```
    short      layer;  
    short      alternateGroup;  
    short      volume;  
    short      reserved4;  
    MatrixRecord matrix;  
    Fixed      trackWidth;  
    Fixed      trackHeight;  
};
```

flags

The high byte is version information; the low byte is a combination of one or more flags (see below).

creationTime

Number of seconds since midnight, January 1, 1904 to the time when the 'trak' (IV-2600) atom was created.

modificationTime

Number of seconds since midnight, January 1, 1904 to the time when the 'trak' atom was last changed.

trackID

The ID number of this track.

reserved1

Reserved; set to 0.

duration

The duration of this track in movie time units.

reserved2

Reserved; set to 0.

reserved3

Reserved; set to 0.

layer

The viewing priority of this track in its movie. QuickTime displays the movie's tracks according to this value; tracks with lower layer numbers are displayed in front and tracks with higher layer numbers are displayed in back.

alternateGroup

A number common to a collection of movie tracks that contain alternate data for one another. QuickTime chooses one track from the group, based on considerations such as localization or hardware capabilities.

volume

The sound volume at which to play this track, ranging from -1.0 to +1.0. The high-order 8 bits contain the integer part; the low-order 8 bits contain the

fractional part. A value of +1.0 constitutes the maximum volume of the user's computer. Negative values are silent but retain the magnitude of the volume setting.

reserved4

Reserved; set to 0.

matrix

A `MatrixRecord` (IV–2304) structure that contains the graphic transformation matrix for this track.

trackWidth

The width of this track in pixels.

trackHeight

The height of this track in pixels.

flags Constants

`TrackEnable`

This track is enabled.

`TrackInMovie`

This track is part of the movie playback.

`TrackInPreview`

This track is part of the movie's **preview**.

`TrackInPoster`

This track is part of the movie's **poster**.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For the atom structure that normally contains the `TrackHeader` data structure, see 'tkhd' (IV–2598).

TrackLoadSettings

Contains preloading information for a track, used in a 'load' (IV–2556) atom.

```
struct TrackLoadSettings {
    TimeValue    preloadStartTime;
    TimeValue    preloadDuration;
    long         preloadFlags;
    long         defaultHints;
};
```

TrackRecord

preloadStartTime
Undocumented
preloadDuration
Undocumented
preloadFlags
Undocumented
defaultHints
Undocumented

Programming Info

C interface file: MoviesFormat.h

See Also

For the atom structure that contains this data structure, see the 'load' (IV–2556) atom.

TrackRecord

Contains a track.

```
struct TrackRecord {  
    long    data[1];  
};
```

data

An array of track data.

Programming Info

C interface file: Movies.h

TuneStatus

Provides information on the currently playing tune.

```
struct TuneStatus {  
    unsigned long *   tune;  
    unsigned long *   tunePtr;  
    TimeValue         time;  
    short              queueCount;  
    short              queueSpots;  
    TimeValue         queueTime;  
    long               reserved[3];  
};
```

};

tune

The currently playing tune.

tunePtr

Current position within the playing tune.

time

Current tune time.

queueCount

Number of tunes queued up.

queueSpots

Number of tunes that can be added to the queue.

queueTime

Total amount of playing time represented by tunes in the queue. This value can be very inaccurate.

reserved

Reserved; set to 0.

Associated Functions

TuneGetStatus (III–1961)

Programming Info

C interface file: QuickTimeMusic.h

TweenRecordPasses information to your **tween** component's TweenDoTween method.

```

struct TweenRecord {
    long            version;
    QTAtomContainer container;
    QTAtom          tweenAtom;
    QTAtom          dataAtom;
    Fixed           percent;
    TweenerDataUPP  dataProc;
    void *          private1;
    void *          private2;
};

```

version

The version number of this structure. This field is initialized to 0.

TweenSequenceEntryRecord

container

The atom container that contains the **tween** data.

tweenAtom

The atom for this **tween** entry's data in the container.

percent

The percentage by which to change the data.

dataProc

The procedure the **tween** component calls to send the tweened value to the receiving track.

private1

Reserved.

private2

Reserved.

Associated Functions

TweenerDataProc (III-2153)

Programming Info

C interface file: Movies.h

TweenSequenceEntryRecord

Specifies when a **tween** in a tween sequence ends.

```
struct TweenSequenceEntryRecord {  
    Fixed      endPercent;  
    QTAtomID   tweenAtomID;  
    QTAtomID   dataAtomID;  
};
```

endPercent

Specifies the point in the duration of the **tween** media sample at which the sequence entry ends. This is expressed as a percentage; for example, if the value is 75.0, the sequence entry ends after three-quarters of the total duration of the tween media sample have elapsed. The sequence entry begins after the end of the previous sequence entry or, for the first entry in the sequence, at the beginning of the tween media sample.

tweenAtomID

Specifies the 'tween' (IV-2605) atom containing the **tween** for the sequence element.

dataAtomID

Specifies the kTweenData atom containing the data for the **tween**.

Discussion

Each **tween** in a tween sequence begins after the previous tween ends or, for the first tween in the sequence, at the beginning of the tween duration. Because there can be more than one data set for a tween, the data structure includes a field for the data atom ID as well as the tween atom ID.

Programming Info

C interface file: `Movies.h`

See Also

See the 'tween' (IV-2605) atom.

TweenV1Record

Provides an extended version of the TweenRecord (IV-2493) structure.

```
struct TweenV1Record {
    long                version;
    QTAtomContainer     container;
    QTAtom              tweenAtom;
    QTAtom              dataAtom;
    Fixed               percent;
    TweenerDataUPP      dataProc;
    void *              private1;
    void *              private2;
    Fract               fractPercent;
};
```

version

The version number of this structure. This field is initialized to 0.

container

The atom container that contains the **tween** data.

tweenAtom

The atom for this **tween** entry's data in the container.

dataAtom

Undocumented

percent

The percentage by which to change the data.

UnsignedWide

dataProc

The procedure the **tween** component calls to send the tweened value to the receiving track.

private1

Reserved.

private2

Reserved.

fractPercent

Undocumented

Programming Info

C interface file: Movies.h

See Also

See the TweenRecord (IV–2493) structure.

UnsignedWide

Stores an unsigned 64-bit value as two unsigned 32-bit integers.

```
struct UnsignedWide {    // big-endian version
    UInt32    hi;
    UInt32    lo;
};
```

```
struct UnsignedWide {    // little-endian version
    UInt32    lo;
    UInt32    hi;
};
```

hi

The high-order unsigned 32-bit integer.

lo

The low-order unsigned 32-bit integer.

Programming Info

C interface file: Endian.h

UserDataRecord

Contains user data.

```
struct UserDataRecord {
    long    data[1];
};
```

data

An array of user data.

Programming Info

C interface file: `Movies.h`

VDCompressionList

Contains a set of image-compression capabilities for a video digitizer.

```
struct VDCompressionList {
    CodecComponent    codec;
    CodecType         cType;
    Str63              typeName;
    Str63              name;
    long               formatFlags;
    long               compressFlags;
    long               reserved;
};
```

codec

Contains the component identifier for the video digitizer's compressor component. Some video digitizers may also implement their image-compression capabilities in an Image Compression Manager compressor component. In this case, the digitizer may allow the application to connect to and use the compressor. If so, the digitizer provides the compressor component's identifier here. If not, the digitizer sets this field to `NIL`.

cType

Identifies the compression algorithm supported by the video digitizer; see "Codec Identifiers" (IV-2655).

typeName

Contains a text string that identifies the compression algorithm. An application may display this string to the user to identify the type of image compression being performed.

name

Specifies the name of the compressor. The developer of the video digitizer assigns this name. An application may display this string to the user.

VDGammaRecord

formatFlags

Contains flags that describe the data formats supported by the video digitizer. Typically, these flags are of interest only to developers of video digitizers and image compressors.

compressFlags

Contains flags that describe the compression capabilities of the video digitizer. Typically, these flags are of interest only to developers of video digitizers and image compressors.

reserved

Reserved; do not use.

Associated Functions

VDGetCompressionTypes (III–1994)

Programming Info

C interface file: QuickTimeComponents.h

VDGammaRecord

Holds a gamma table.

```
struct VDGammaRecord {  
    Ptr    csGTable;  
};
```

csGTable

A pointer to a gamma table.

Programming Info

C interface file: Video.h

VdigBufferRec

Defines a video digitizer buffer.

```
struct VdigBufferRec {  
    PixMapHandle  dest;  
    Point         location;  
    long          reserved;  
};
```


`dest`

Contains a handle to the pixel map that defines the destination buffer.

`location`

Specifies the location of the video destination in the pixel map specified by the `dest` field. This point identifies the upper-left corner of the destination rectangle. The size and scaling of the destination rectangle are governed by the `matrix` and `mask` fields of the buffer list structure that contains this structure.

`reserved`

Reserved; set to 0.

Programming Info

C interface file: `QuickTimeComponents.h`

See Also

See `VdigBufferRecList` (IV–2499).

VdigBufferRecList

Contains a list of `VdigBufferRec` (IV–2498) structures.

```
struct VdigBufferRecList {
    short          count;
    MatrixRecordPtr matrix;
    RgnHandle      mask;
    VdigBufferRec  list[1];
};
```

`count`

Specifies the number of buffers defined by this structure. The value of this field must correspond to the number of entries in the list array.

`matrix`

Specifies the transformation matrix that is applied to all of the destination rectangles before the video image is displayed. You must specify a matrix. If you do not want to perform any transformations, use the identity matrix.

`mask`

Specifies a clipping region that is applied to the destination rectangle before the video image is displayed. Note that this region applies to only the first destination buffer. If you want the region to apply to all of your destination buffers, you must do this yourself. If you do not want to specify a clipping region, set this field to `NIL`.

VdigType

list

Contains an array of VdigBufferRec (IV–2498) structures. Each buffer is represented by a buffer structure.

Associated Functions

VDSetupBuffers (III–2055)

Programming Info

C interface file: QuickTimeComponents.h

See Also

See VdigBufferRec (IV–2498).

VdigType

Defines a video digitizer's type.

```
struct VdigType {  
    long    digType;  
    long    reserved;  
};
```

digType

A constant (see below) that defines a video digitizer type.

reserved

Reserved; do not use.

digType Constants

vdTypeBasic

Basic video digitizer; does not support any clipping.

vdTypeAlpha

Supports clipping by means of an alpha channel.

vdTypeMask

Supports clipping by means of a mask plane.

vdTypeKey

Supports clipping by means of key colors.

Programming Info

C interface file: QuickTimeComponents.h

See Also

See VdigTypeList (IV–2501).

VdigTypeList

Contains a list of VdigType (IV–2500) structures.

```
struct VdigTypeList {
    short      count;
    VdigType   list[1];
};
```

count

Number of structures in the list.

list

An array of VdigType (IV–2500) structures.

Programming Info

C interface file: QuickTimeComponents.h

See Also

See VdigType (IV–2500).

VideoBottles

Identifies the callback functions to be assigned to a sequence grabber channel.

```
struct VideoBottles {
    short      procCount;
    SGGrabBottleUPP      grabProc;
    SGGrabCompleteBottleUPP      grabCompleteProc;
    SGDisplayBottleUPP      displayProc;
    SGCompressBottleUPP      compressProc;
    SGCompressCompleteBottleUPP      compressCompleteProc;
    SGAddFrameBottleUPP      addFrameProc;
    SGTransferFrameBottleUPP      transferFrameProc;
    SGGrabCompressCompleteBottleUPP      grabCompressCompleteProc;
    SGDisplayCompressBottleUPP      displayCompressProc;
};
```

procCount

Specifies the number of callback functions that may be identified in the structure. Set this field to 9.

grabProc

Identifies the SGGrabBottleProc (III–2142) function. If you are setting such a function, set this field so that it points to the function's entry point. If you are not setting such a function, set this field to NIL.

VideoBottles

grabCompleteProc

Identifies the `SGGrabCompleteBottleProc` (III–2143) function. If you are setting such a function, set this field so that it points to the function’s entry point. If you are not setting such a function, set this field to `NIL`.

displayProc

Identifies the `SGDisplayBottleProc` (III–2140) function. If you are setting such a function, set this field so that it points to the function’s entry point. If you are not setting such a function, set this field to `NIL`.

compressProc

Identifies the `SGCompressBottleProc` (III–2137) function. If you are setting such a function, set this field so that it points to the function’s entry point. If you are not setting such a function, set this field to `NIL`.

compressCompleteProc

Identifies the `SGCompressCompleteBottleProc` (III–2138) function. If you are setting such a function, set this field so that it points to the function’s entry point. If you are not setting such a function, set this field to `NIL`.

addFrameProc

Identifies the `SGAddFrameBottleProc` (III–2136) function. If you are setting such a function, set this field so that it points to the function’s entry point. If you are not setting such a function, set this field to `NIL`.

transferFrameProc

Identifies the `SGTransferFrameBottleProc` (III–2145) function. If you are setting such a function, set this field so that it points to the function’s entry point. If you are not setting such a function, set this field to `NIL`.

grabCompressCompleteProc

Identifies the `SGGrabCompressCompleteBottleProc` (III–2143) function. If you are setting such a function, set this field so that it points to the function’s entry point. If you are not setting such a function, set this field to `NIL`.

displayCompressProc

Identifies the `SGDisplayCompressBottleProc` (III–2141) function. If you are setting such a function, set this field so that it points to the function’s entry point. If you are not setting such a function, set this field to `NIL`.

Associated Functions

`SGGetVideoBottlenecks` (III–1741)

Programming Info

C interface file: `QuickTimeComponents.h`

VideoMediaInfoHeader

Defines the graphics transfer characteristics for a video media.

```
struct VideoMediaInfoHeader {
    long    flags;
    short   graphicsMode;
    short   opColorRed;
    short   opColorGreen;
    short   opColorBlue;
};
```

flags

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

graphicsMode

A transfer mode constant; see “Graphics Transfer Modes” (IV–2670).

opColorRed

Red parameter to be passed to the transfer operation.

opColorGreen

Green parameter to be passed to the transfer operation.

opColorBlue

Blue parameter to be passed to the transfer operation.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For the atom structure that contains `VideoMediaInfoHeader`, see `'vmhd'[media]` (IV–2611).

wide

Stores a signed 64-bit value as a signed 32-bit integer and an unsigned 32-bit integer.

```
struct wide {           // big-endian version
    SInt32    hi;
    UInt32    lo;
};

struct wide {           // little-endian version
    UInt32    lo;
    SInt32    hi;
};
```

XMLAttribute

```
};
```

```
hi
```

The signed high-order 32-bit integer.

```
lo
```

The unsigned low-order 32-bit integer.

Programming Info

C interface file: `Endian.h`

See Also

See the `UnsignedWide` (IV–2496) structure.

XMLAttribute

An XML attribute-value pair

```
struct XMLAttribute {  
    UInt32          identifier;  
    char *          name;  
    long            valueKind;  
    XMLAttributeValue value;  
    char *          valueStr;  
};
```

`identifier`

A tokenized identifier, if the attribute name was recognized by the parser.

`name`

The attribute's name. This is present only if the `identifier` parameter has the value `xmlIdentifierUnrecognized`.

`valueKind`

The type of the parsed value, if the value was recognized and parsed; otherwise, the value `attributeValueKindCharString`.

`value`

The parsed attribute value.

`valueStr`

The value string.

Programming Info

C interface file: `QuickTimeComponents.h`

XMLContent

Undocumented

```
struct XMLContent {
    UInt32          kind;
    XMLElementContent actualContent;
};
```

kind

Undocumented

actualContent

Undocumented

Programming Info

C interface file: QuickTimeComponents.h

XMLDocRecord

Undocumented

```
struct XMLDocRecord {
    void *          xmlDataStorage;
    XMLElement      rootElement;
};
```

xmlDataStorage

Undocumented

rootElement

Undocumented

Programming Info

C interface file: QuickTimeComponents.h

XMLElement

Undocumented

```
struct XMLElement {
    UInt32          identifier;
    char *          name;
    XMLAttributePtr attributes;
```

XMLElement

```
XMLContentPtr    contents;  
};
```

identifier

A tokenized identifier, if the element name was recognized by the parser.

name

The element name, only present if the identifier parameter contains
xmlIdentifierUnrecognized.

attributes

An array of attributes, terminated with a value of xmlIdentifierInvalid.

contents

Array of contents, terminated with a value of xmlIdentifierInvalid

Programming Info

C interface file: QuickTimeComponents.h

Atoms

0x00000000

Terminates an audio atom list.

```
struct AudioTerminatorAtom {
    long      size;
    OSType    atomType;
};
```

size

The size in bytes of this atom structure.

atomType

Constant kAudioTerminatorAtomType; see “Atom ID Codes” (IV–2649).

Programming Info

Sound.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

0x00000001

A **sprite** property matrix atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

```
atomType
    0x00000001.
```

data

The **sprite** matrix property, a structure of type MatrixRecord (IV–2304).

Parent Atom

'sprt' (IV–2584)

Parent atom can contain only one atom of this type.

0x00000004

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

0x00000004

A **sprite** visible property atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II-1183) using the following parameters:

atomType
0x00000004.

data
The **sprite** visible property, of type short.

Parent Atom

'sprt' (IV-2584)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

0x00000005

A **sprite** property layer atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II-1183) using the following parameters:

atomType
0x00000005.

data
The **sprite** layer property, of type short.

Parent Atom

'sprt' (IV-2584)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

0x00000006

A **sprite** graphics mode property atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

```
atomType
    0x00000006.
```

```
data
    The sprite graphics mode property, a structure of type
    ModifierTrackGraphicsModeRecord (IV–2311).
```

Parent Atom

```
'sprt' (IV–2584)
    Parent atom can contain only one atom of this type.
```

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

0x00000064

A **sprite** image index atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

```
atomType
    0x00000064.
```

```
data
    The sprite image index property, of type short.
```

Parent Atom

```
'sprt' (IV–2584)
    Parent atom can contain only one atom of this type.
```

0x00000065

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

0x00000065

A **sprite** background color property atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType
0x00000065.

data
The **sprite** background color property, a structure of type RGBColor (IV–2403).

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

0x00000066

A **sprite** property offscreen bit depth atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType
0x00000066.

data
The **sprite** offscreen bit depth property, of type short.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

0x00000067

A **sprite** property sample format atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

```
atomType
    0x00000067.
```

```
data
```

The **sprite** sample format property, of type short.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'A11F'

User data list entry atom to play all frames.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

```
size
```

The size in bytes of this atom structure.

```
udType
```

```
'A11F'.
```

```
data
```

A byte indicating that all frames of video should be played, regardless of timing.

Parent Atom

```
'udta' (IV-2607)
```

Parent atom can contain only one atom of this type.

Programming Info

```
MoviesFormat.h
```

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'beha'

'beha'

Defines **sprite** behavior.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

atomType
Value is 'beha'.

Optional Child Atoms

'imag' (IV–2547)

A **sprite** image atom.

'cusr' (IV–2524)

Color custom cursor child atom.

'sstr' (IV–2587)

Specifies the ID of a string variable, contained in a **sprite** track, to display in the status area of the browser.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'chap'

Chapter or scene list track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType
Constant `kTrackReferenceChapterList`, designating atom type 'chap'; see “Atom ID Codes” (IV–2649).

data

A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

'tref' (IV–2602)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'clip'

Defines a clipping region.

```
struct ClippingAtom {
    long        size;
    long        atomType;
    RgnAtom     aRgnClip;
};
```

size

The size in bytes of this atom structure.

atomType

Constant ClipAID, designating atom type 'clip'; see “Atom ID Codes” (IV–2649).

aRgnClip

A 'crgn' (IV–2523) atom that defines the clipping region.

Discussion

You can treat this atom either as a declared structure or as a QT atom, which you can create it with QTInsertChild (II–1183).

Programming Info

MoviesFormat.h

See Also

For the atoms that may contain this atom, see 'moov' (IV–2566) and 'trak' (IV–2600). For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'clon'

Contains information about a track clone.

```
struct CloneAtom {
    long        size;
    long        atomType;
    CloneRecord cloneInfo;
};
```

'cmov'

size

The size in bytes of this atom structure.

atomType

Value is 'clon'.

cloneInfo

A CloneRecord (api>CloneRecord–CloneRecord) structure.

See Also

See the CloneRecord (api>CloneRecord–CloneRecord) structure and AddClonedTrackToMovie (I–22). For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'cmov'

Contains a compressed movie.

This is a QT atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameter:

atomType

Constant CompressedMovieAID, designating atom type 'cmov'; see “Atom ID Codes” (IV–2649).

Parent Atom

'moov' (IV–2566)

Parent atom can contain only one atom of this type.

Required Child Atoms

'dcom' (IV–2527)

Indicates the compression algorithm used to compress a movie. Only one allowed.

'cmvd' (IV–2514)

Stores the data for a compressed movie. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'cmvd'

Stores the data for a compressed movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

`atomType`

Constant `CompressedMovieDataAID`, designating atom type 'cmvd'; see “Atom ID Codes” (IV-2649).

`data`

An integer of type `UInt32` that gives the length of the uncompressed movie in bytes, followed by the compressed movie data.

Parent Atom

'cmov' (IV-2514)

Parent atom can contain only one atom of this type.

'co64'

A 64-bit version of the 'stco' (IV-2589) atom.

For details, see 'stco' (IV-2589).

`atomType`

Constant `STChunkOffset64AID`, designating atom type 'co64'; see “Atom ID Codes” (IV-2649).

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'©cpy'

User data list entry atom: copyright information.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

`size`

The size in bytes of this atom structure.

'©day'

udType

Constant kUserDataTextCopyright, designating atom type '@cpy'; see “User Data Identifiers” (IV–2696).

data

A string containing copyright information.

Parent Atom

'udta' (IV-2607)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'©day'

User data list entry atom: creation date.

```
struct MoviesUserData {  
    long    size;  
    long    udType;  
    char    data[1];  
};
```

size

The size in bytes of this atom structure.

udType

Constant kUserDataTextCreationDate, designating atom type '@day'; see “User Data Identifiers” (IV–2696).

data

A string containing the creation date.

Parent Atom

'udta' (IV-2607)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'©dir'

User data list entry atom: name of movie's director.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

size

The size in bytes of this atom structure.

udType

Constant `kUserDataTextDirector`, designating atom type '`©dir`'; see “User Data Identifiers” (IV–2696).

data

A string containing the name of the movie's director.

Parent Atom

'udta' (IV–2607)

Parent atom can contain only one atom of this type.

Programming Info

`MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'©ed1'

User data list entry atom: edit date 1.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

'©fmt'

size

The size in bytes of this atom structure.

udType

Constant `kUserDataTextEditDate1`, designating atom type '@ed1'; see “User Data Identifiers” (IV–2696).

data

A string containing the first edit date.

Parent Atom

'udta' (IV-2607)

Parent atom can contain only one atom of this type.

Discussion

Similar atoms of types '@ed2' through '@ed9' may contain other edit date strings.

Programming Info

`MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'©fmt'

User data list entry atom: indication of movie’s format.

```
struct MoviesUserData {  
    long    size;  
    long    udType;  
    char    data[1];  
};
```

size

The size in bytes of this atom structure.

udType

Constant `kUserDataTextOriginalFormat`, designating atom type '@fmt'; see “User Data Identifiers” (IV–2696).

data

A string indicating the movie’s format.

Parent Atom

'udta' (IV-2607)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'©inf'

User data list entry atom: information about the movie.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

size

The size in bytes of this atom structure.

udType

Constant kUserDataTextInformation, designating atom type '©inf'; see “User Data Identifiers” (IV-2696).

data

A string containing information about the movie.

Parent Atom

'udta' (IV-2607)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'©prd'

User data list entry atom: name of movie's producer.

```
struct MoviesUserData {
    long    size;
    long    udType;
```

'©prf'

```
char    data[1];  
};
```

size

The size in bytes of this atom structure.

udType

Constant `kUserDataTextProducer`, designating atom type `'©prd'`; see “User Data Identifiers” (IV–2696).

data

A string containing the name of the movie’s producer.

Parent Atom

`'udta'` (IV–2607)

Parent atom can contain only one atom of this type.

Programming Info

`MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'©prf'

User data list entry atom: names of performers.

```
struct MoviesUserData {  
    long    size;  
    long    udType;  
    char    data[1];  
};
```

size

The size in bytes of this atom structure.

udType

Constant `kUserDataTextPerformers`, designating atom type `'©prf'`; see “User Data Identifiers” (IV–2696).

data

A string containing names of the performers.

Parent Atom

`'udta'` (IV–2607)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'©req'

User data list entry atom: special hardware or software requirements.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

size

The size in bytes of this atom structure.

udType

Constant kUserDataTextSpecialPlaybackRequirements, designating atom type '©req'; see “User Data Identifiers” (IV–2696).

data

A string detailing special hardware or software requirements.

Parent Atom

'udta' (IV–2607)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'©src'

User data list entry atom: credits for those who provided movie source content.

```
struct MoviesUserData {
    long    size;
    long    udType;
```

'©wrt'

```
char    data[1];  
};
```

size

The size in bytes of this atom structure.

udType

Constant `kUserDataTextOriginalSource`, designating atom type '@src'; see “User Data Identifiers” (IV–2696).

data

A string containing credits for those who provided movie source content.

Parent Atom

'udta' (IV–2607)

Parent atom can contain only one atom of this type.

Programming Info

`MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'©wrt'

User data list entry atom: name of movie’s writer.

```
struct MoviesUserData {  
    long    size;  
    long    udType;  
    char    data[1];  
};
```

size

The size in bytes of this atom structure.

udType

Constant `kUserDataTextWriter`, designating atom type '@wrt'; see “User Data Identifiers” (IV–2696).

data

A string containing the name of the movie’s writer.

Parent Atom

'udta' (IV–2607)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'crgn'

Defines a clipping region.

```
struct RgnAtom {
    long    size;
    long    atomType;
    short   rgnSize;
    Rect    rgnBBox;
    char    data[1];
};
```

size

The size in bytes of this atom structure.

atomType

Constant `RgnClipAID`, designating atom type 'crgn'; see “Atom ID Codes” (IV–2649).

rgnSize

The size in bytes of the region.

rgnBBox

The bounding box for the region.

data

Additional data if the clipping region is not rectangular.

Parent Atom

'clip' (IV-2513)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'crsr'

'crsr'

Color custom cursor child atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `kSpriteCursorBehaviorAtomType`, designating atom type 'crsr'; see “Atom ID Codes” (IV–2649).

data

A cursor description.

Parent Atom

'vrcp' (IV-2612)

Parent atom can contain multiple atoms of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'cspd'

Contains the connection speed currently set in the QuickTime preferences.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `ConnectionSpeedPrefsType`, designating atom type 'cspd'; see “Atom ID Codes” (IV–2649).

data

The connection speed.

See Also

See the `GetQuickTimePreference` (I–507) and `SetQuickTimePreference` (III–1639) functions. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'ctab'

Color table atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `ColorTableAID`, designating atom type 'ctab'; see “Atom ID Codes” (IV–2649).

`data`

A color table.

Parent Atom

'moov' (IV–2566)

Parent atom can contain only one atom of this type.

Discussion

Color table atoms define a list of preferred colors for displaying the movie on devices that support only 256 colors. The list may contain up to 256 colors.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'cufa'

Non-standard cubic QTVR panorama data atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Value is 'cufa'.

`data`

A `QTVRCubicFaceData` (IV–2393) structure.

Discussion

Each entry in the `QTVRCubicFaceData` structure describes one face of the polyhedron being described.

'CURS'

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'CURS'

Custom cursor child atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `kQTVRCursorAtomType`, designating atom type 'CURS'; see “Atom ID Codes” (IV–2649).

data

A cursor description.

Parent Atom

'vrcp' (IV–2612)

Parent atom can contain multiple atoms of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'cuvw'

Cubic view atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Value is 'cuvw'.

data

A `QTVRCubicViewAtom` (IV–2394) structure.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'dasz'

Data size atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `kQTTargetDataSize`, designating atom type 'dasz'; see “Atom ID Codes” (IV–2649).

`data`

A `QTTargetDataSize` (IV–2391) structure.

Parent Atom

'vide' (IV–2610)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'dcom'

Indicates the compression algorithm used to compress a movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `DataCompressionAtomAID`, designating atom type 'dcom'; see “Atom ID Codes” (IV–2649).

`data`

A 32-bit constant (see below) that indicates which lossless algorithm was used to compress the movie contained in the parent atom.

Data Constants

`AppleDataCompressorSubType`

The Apple data compressor; value is 'adec'.

`zlibDataCompressorSubType`

The zlib data compressor; value is 'zlib'.

'defi'

Parent Atom

'cmov' (IV-2514)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'defi'

A **sprite** image data reference atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

atomType

Constant `kSpriteImageDefaultImageIndexAtomType`, designating atom type 'defi'; see “Atom ID Codes” (IV-2649).

data

The image index of a traditional image, of type short, to use while waiting for the referenced image to load.

Parent Atom

'imag' (IV-2547)

Parent atom can contain only one atom of this type.

Discussion

You use the this atom type to specify that an image is referenced and how to access it. Its ID should be 1.

Version Notes

Added to QuickTime 4.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'desc'

Graphics export description atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

atomType

Constant kCustomHandlerDesc, designating atom type 'desc'; see “Atom ID Codes” (IV–2649).

data

A nonterminated string containing a human-readable format name.

Parent Atom

'expo' (IV–2535)

Parent atom can contain only one atom of this type.

See Also

See the GraphicsExportGetMIMETypeList (I–577) and GraphicsImportGetExportImageTypeList (I–632) functions. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'dflt'

Key frame shared-data atom.

This is a QT atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameter:

atomType

Constant kSpriteSharedDataAtomType, designating atom type 'dflt'; see “Atom ID Codes” (IV–2649).

Required Child Atoms

'imct' (IV–2549)

Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'dimmm'

Number of bytes of immediate data to be sent.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

'dinf'

atomType

Value is 'dimm'.

data

8-byte value.

Parent Atom

'hinf' (IV-2541)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'dinf'

Specifies where media data is stored.

```
struct DataInfoAtom {  
    long          size;  
    long          atomType;  
    DataRefAtom   dataRef;  
};
```

size

The size in bytes of this atom structure.

atomType

Constant DataInfoAID, designating atom type 'dinf'; see “Atom ID Codes” (IV-2649).

dataRef

A value that contains the data for this atom. The 4-byte DataRefAtom data type is private and is not documented.

Optional Child Atoms

'dref' (IV-2532)

Only one allowed.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'dmax'

The largest packet duration.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Value is 'dmax'.

data

4 bytes packet duration, in milliseconds.

Parent Atom

'hinf' (IV–2541)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'dmed'

Number of bytes from the media track to be sent.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Value is 'dmed'.

data

8-byte value.

Parent Atom

'hinf' (IV–2541)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'dref'

'dref'

Data reference atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `DataRefAID`, designating atom type 'dref'; see “Atom ID Codes” (IV–2649).

`data`

Data references.

Parent Atom

'dinf' (IV–2530)

Parent atom can contain only one atom of this type.

See Also

See the `AliasRecord` (IV–2159) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'drep'

Number of bytes of repeated data to be sent.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Value is 'drep'.

`data`

8-byte value.

Parent Atom

'hinf' (IV–2541)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'edts'

Contains an atom that defines an edit list.

```
struct EditsAtom {
    long          size;
    long          atomType;
    EditListAtom  editList;
};
```

size

The size in bytes of this atom structure.

atomType

Constant EditsAID, designating atom type 'edts'; see “Atom ID Codes” (IV–2649).

editList

An 'elst' (IV–2533) atom.

Parent Atom

'trak' (IV–2600)

Parent atom can contain only one atom of this type.

Required Child Atoms

'elst' (IV–2533)

Only one allowed.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'elst'

Contains a list of edit segment definitions for a media.

```
struct EditListAtom {
    long          size;
    long          atomType;
    long          flags;
    long          numEntries;
    EditListType  editListTable[1];
};
```

'end '

size

The size in bytes of this atom structure.

atomType

Constant `EditListAID`, designating atom type 'elst'; see “Atom ID Codes” (IV–2649).

flags

One byte of version information followed by three bytes of flags. The flag bytes are not currently used.

numEntries

The number of entries in `editListTable`.

editListTable

An array of `EditListType` (IV–2241) data structures, each of which locates and defines an edit segment within a media.

Parent Atom

'edts' (IV–2533)

Parent atom can contain only one atom of this type.

Discussion

You can use the edit list atom to tell QuickTime how to map from a time in a movie to a time in a media, and ultimately to each segment of the media’s data.

Programming Info

`MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'end '

Defines the ending offset of hypertext in a text stream.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Value is 'end ' (the fourth character is a space).

data

The ending offset of hypertext in a text stream.

Parent Atom

'htxt' (IV-2546)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'enda'

Determines the endian status of the sound component that interprets data contained in an audio atom list.

```
struct AudioEndianAtom {
    long    size;
    OSType  atomType;
    short    littleEndian;
};
```

size

The size in bytes of this atom structure.

atomType

Constant kAudioEndianAtomType, designating atom type 'enda'; see “Atom ID Codes” (IV-2649).

littleEndian

Set this field to TRUE if the audio component is to operate on **little-endian** data, and FALSE otherwise.

Programming Info

Sound.h

See Also

To choose the sound component for an audio atom list, see the 'frma' (IV-2538) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'expo'

Defines a graphics export group.

'ext '

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameters:

`atomType`

Constant `kGraphicsExportGroup`, designating atom type 'expo'; see “Atom ID Codes” (IV–2649).

Required Child Atoms

'ftyp' (IV-2539)

An OSType representing the exported file type.

'ext ' (IV-2536)

A nonterminated string containing the suggested file extension for this format.

'desc' (IV-2528)

A nonterminated string containing a human-readable name for this format.

Optional Child Atoms

'mime' (IV-2561)

A nonterminated string containing the MIME type for this format.

See Also

See the `GraphicsImportGetExportImageTypeList` (I–632) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'ext '

Defines a graphics export extension.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `kGraphicsExportExtension`, designating atom type 'ext ' (the fourth character is a space); see “Atom ID Codes” (IV–2649).

`data`

A nonterminated string containing a file extension.

Parent Atom

'expo' (IV-2535)

Parent atom can contain only one atom of this type.

See Also

See the `GraphicsImportGetExportImageTypeList` (I–632) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'flap'

Extension to the `SoundDescription` (IV–2449) structure.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `siSlopeAndIntercept`, designating atom type 'flap'; see “Sound Information Selectors” (IV–2693).

`data`

A `SoundSlopeAndInterceptRecord` (IV–2456) structure.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'flov'

Contains a floating-point variable for a **sprite**.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

`atomType`

Value is 'flov'.

Parent Atom

'vars' (IV–2610)

Contains variables for a **sprite**.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'free'

'free'

Provides unused space in a movie file.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `FreeAtomType`, designating atom type 'free'; see “Atom ID Codes” (IV–2649).

data

Any number of bytes of free space.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'frma'

Specifies which sound component is responsible for the atoms contained in an audio atom list.

```
struct AudioFormatAtom {  
    long    size;  
    OSType  atomType;  
    OSType  format;  
};
```

size

The size in bytes of this atom structure.

atomType

Constant `kAudioFormatAtomType`, designating atom type 'frma'; see “Atom ID Codes” (IV–2649).

format

A constant that identifies a sound component. See “Codec Identifiers” (IV–2655).

Programming Info

`Sound.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'ftyp'

Defines a graphics export file type.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `kGraphicsExportFileType`, designating atom type 'ftyp'; see “Atom ID Codes” (IV–2649).

`data`

An `OSType` representing the exported file type.

Parent Atom

'expo' (IV–2535)

Parent atom can contain only one atom of this type.

See Also

See the `GraphicsImportGetExportImageTypeList` (I–632) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'gmhd'

Contains a generic media information atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

`atomType`

Constant `GenericMediaInfoHeaderAID`, designating atom type 'gmhd'; see “Atom ID Codes” (IV–2649).

Parent Atom

'minf'[generic] (IV–2562)

Parent atom can contain only one atom of this type.

'gmin'

Required Child Atoms

'gmin' (IV-2540)

Provides data that is specific to a handler for media other than video or sound.
Only one allowed.

Discussion

This atom is currently used only as a container for a 'gmin' (IV-2540) atom.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'gmin'

Provides data that is specific to a handler for media other than video or sound.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

`atomType`

Constant `GenericMediaInfoAID`, designating atom type 'gmin'; see “Atom ID Codes” (IV-2649).

`data`

Data required by the media handler that is designated by the 'hdlr' (IV-2540) atom contained in the 'minf'[generic] (IV-2562) atom that also contains the parent of this atom.

Parent Atom

'gmhd' (IV-2539)

Parent atom can contain any number of atoms of this type.

Discussion

This atom contains handler-specific information to support your use of a 'minf'[generic] (IV-2562) atom. Note that the data in this atom is not used by RTP servers.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'hdlr'

Specifies the component that is to interpret a media's data.

```
struct HandlerAtom {
    long          size;
    long          atomType;
    PublicHandlerInfo hInfo;
};
```

size

The size in bytes of this atom structure.

atomType

Constant HandlerAID, designating atom type 'hdlr'; see “Atom ID Codes” (IV–2649).

hInfo

A PublicHandlerInfo (IV–2341) structure, which contains the actual data for this atom.

Discussion

RTP servers ignore this atom’s data when it is contained in a 'minf'[generic] (IV–2562) atom.

Programming Info

MoviesFormat.h

See Also

For the atoms that may contain this atom, see 'mdia' (IV–2560), 'minf'[generic] (IV–2562), 'minf'[sound] (IV–2564), and 'minf'[video] (IV–2565). For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'hinf'

Contains statistics for the hint track.

This is a QT container atom; it is not declared in the header files. You can create it with QTNewAtomContainer (II–1203) and insert it with QTInsertChild (II–1183), using the following parameter:

atomType

Value is 'hinf'.

Required Child Atoms

'trpy' (IV–2603)

Total number of bytes that will be sent, including 12-byte RTP headers but not including network headers. Only one allowed.

'hint'

'nump' (IV-2570)

Total number of network packets that will be sent. Only one allowed.

'tpyl' (IV-2600)

Total number of bytes that will be sent, not including 12-byte RTP headers.
Only one allowed.

'maxr' (IV-2558)

Maximum data rate. Only one allowed.

'dmed' (IV-2531)

Number of bytes from the media track to be sent. Only one allowed.

'dimm' (IV-2529)

Number of bytes of immediate data to be sent. Only one allowed.

'drep' (IV-2532)

Number of bytes of repeated data to be sent. Only one allowed.

'tmin' (IV-2599)

Smallest relative transmission time, in milliseconds. Only one allowed.

'tmax' (IV-2598)

Largest relative transmission time, in milliseconds. Only one allowed.

'pmax' (IV-2572)

Largest packet, in bytes, including 12-byte RTP header. Only one allowed.

'dmax' (IV-2531)

The largest packet duration. Only one allowed.

'payt' (IV-2571)

Payload type. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'hint'

Hint track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II-1183) using the following parameters:

atomType

Value is 'hint'.

data

A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

'tref' (IV-2602)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'hlit'

Defines the highlighted portion in text.

```
struct HiliteAtom {  
    long    size;  
    long    atomType;  
    long    selStart;  
    long    selEnd;  
};
```

size

The size in bytes of this atom structure.

atomType

Value is 'hlit'.

selStart

Character number of highlighted selection start character.

selEnd

Character number of highlighted selection end character.

Discussion

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

Programming Info

MoviesFormat.h

'hnti'

'hnti'

Hint track user data atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

```
atomType
    Value is 'hnti'.
```

Required Child Atoms

```
'sdp ' (IV–2582)
    SDP text for a hint track. Only one allowed.
```

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'hots'

A QTVR hot spot.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

```
atomType
    Constant kQTVRHotSpotAtomType, designating atom type 'hots'; see “Atom ID Codes” (IV–2649).
```

Parent Atom

```
'hspace' (IV–2545)
    Parent atom can contain any number of atoms of this type.
```

Required Child Atoms

```
'hsin' (IV–2545)
    Contains general hot spot information. Only one allowed.
'link' (IV–2556)
    Specific information about a link hot spot. Only one allowed.
```

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'hsin'

Contains general hot spot information.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `kQTVRHotSpotInfoAtomType`, designating atom type 'hsin'; see “Atom ID Codes” (IV–2649).

`data`

Hot spot information.

Parent Atom

'hots' (IV–2544)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'hspa'

Hot spot parent atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

`atomType`

Constant `kQTVRHotSpotParentAtomType`, designating atom type 'hspa'; see “Atom ID Codes” (IV–2649).

Parent Atom

'vrnp' (IV–2614)

Parent atom can contain only one atom of this type.

Required Child Atoms

'hots' (IV–2544)

A QTVR hot spot. Any number allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'htxt'

'htxt'

Hypertext in a text stream.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

atomType
Value is 'htxt'.

Parent Atom

'wtxt' (IV–2616)
Parent atom can contain multiple atoms of this type.

Required Child Atoms

'strt' (IV–2589)
Defines the starting offset of hypertext in a text stream. Only one allowed.
'end ' (IV–2534)
Defines the ending offset of hypertext in a text stream. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'idat'

Image data atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType
Constant `quickTimeImageFileImageDataAtom`, designating atom type 'idat'; see “Atom ID Codes” (IV–2649).
data
The image data.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'idsc'

Image description atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `quickTimeImageFileImageDescriptionAtom`, designating atom type `'idsc'`; see “Atom ID Codes” (IV–2649).

`data`

The image description.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'iicc'

ColorSync profile atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `quickTimeImageFileColorSyncProfileAtom`, designating atom type `'iicc'`; see “Atom ID Codes” (IV–2649).

`data`

A ColorSync profile.

Version Notes

This is a new optional atom in QuickTime 4.

See Also

See the `GraphicsExportSetColorSyncProfile` (I–589) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'imag'

A **sprite** image atom.

'imap'

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

`atomType`

Constant `kSpriteImageAtomType`, designating atom type 'imap'; see “Atom ID Codes” (IV–2649).

Parent Atom

'imct' (IV–2549)

Parent atom can contain any number of atoms of this type.

Required Child Atoms

'imda' (IV–2549)

Contains **sprite** image data. Only one allowed.

Optional Child Atoms

'name'[sprite] (IV–2568)

A **sprite** name atom. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'imap'

An input map.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

`atomType`

Constant `InputMapAID`, designating atom type 'imap'; see “Atom ID Codes” (IV–2649).

Parent Atom

'trak' (IV–2600)

Parent atom can contain only one atom of this type.

Required Child Atoms

' in' (IV–2554)

Track input atom. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'imct'

A **sprite** image container atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

atomType

Constant `kSpriteImagesContainerAtomType`, designating atom type 'imct'; see “Atom ID Codes” (IV–2649).

Parent Atom

'df1t' (IV–2529)

Parent atom can contain only one atom of this type.

Required Child Atoms

'imag' (IV–2547)

Sprite image atom. Any number allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'imda'

A **sprite** image data.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `kSpriteImageDataAtomType`, designating atom type 'imda'; see “Atom ID Codes” (IV–2649).

data

Image data.

'imgp'

Parent Atom

'imag' (IV-2547)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'imgp'

Panorama imaging parent atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II-1203) and insert it with `QTInsertChild` (II-1183), using the following parameter:

atomType

Constant `kQTVRImagingParentAtomType`, designating atom type 'imgp'; see “Atom ID Codes” (IV-2649).

Required Child Atoms

'impn' (IV-2551)

Panorama imaging atom. Any number allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'imgr'

A **sprite** image group ID atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

atomType

Constant `kSpriteImageGroupIDAtomType`, designating atom type 'imgr'; see “Atom ID Codes” (IV-2649).

data

The group ID, of type long.

Parent Atom

'imag' (IV-2547)

Parent atom can contain only one atom of this type.

Discussion

Each image in a **sprite** media **key frame** sample is assigned to a group. Add an atom of this type as a child of the 'imag' (IV-2547) atom and set its leaf data to a long containing the group ID. For example, if the sample contains ten images where the first two images are equivalent, and the last eight images are equivalent, then you could assign a group ID of 1000 to the first two images, and a group ID of 1001 to the last eight images. This divides the images in the sample into two sets. The actual ID does not matter, it just needs to be a unique positive integer. Note that you must assign group IDs to your sprite sample if you want a sprite to display images with non-equivalent image descriptions (i.e., images with different dimensions).

Special Considerations

Although QuickTime does not currently use this atom internally, tools that edit **sprite** media can use the information provided to optimize certain operations, such as cut, copy, and paste.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'impn'

Panorama imaging atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II-1183) using the following parameters:

atomType

Constant kQTVRPanoImagingAtomType, designating atom type 'impn'; see “Atom ID Codes” (IV-2649).

data

A QTVRPanoImagingAtom (IV-2398) structure.

Parent Atom

'imgp' (IV-2550)

Parent atom can contain any number of atoms of this type.

'imre'

Discussion

A `QTVRPanoImagingAtom` describes the default imaging characteristics for all the panoramic nodes in a scene. This atom overrides QuickTime VR's own defaults.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'imre'

A **sprite** image data reference atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

`atomType`

Constant `kSpriteImageDataRefAtomType`, designating atom type 'imre'; see “Atom ID Codes” (IV-2649).

`data`

The data reference, which is similar to the `dataRef` parameter of `GetDataHandler` (I-407).

Parent Atom

'imag' (IV-2547)

Parent atom can contain only one atom of this type.

Discussion

You use the this atom type to specify that an image is referenced and how to access it. Add this atom as a child of the 'imag' (IV-2547) atom instead of an 'imda' (IV-2549) atom. Its ID should be 1.

Version Notes

Added in QuickTime 4.

See Also

See the `GetDataHandler` (I-407) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'imrg'

Custom **sprite** image registration point atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `kSpriteImageRegistrationAtomType`, designating atom type 'img'; see “Atom ID Codes” (IV–2649).

`data`

The desired **sprite** registration point, a `FixedPoint` (IV–2258) structure.

Parent Atom

'img' (IV–2547)

Parent atom can contain only one atom of this type.

Discussion

Sprite images have a default registration point of 0, 0. To specify a different point, add an atom of this type as a child atom of the 'img' (IV–2547) and set its leaf data to a `FixedPoint` value with the desired registration point.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'imrt'

A **sprite** image data reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `kSpriteImageDataRefTypeAtomType`, designating atom type 'imrt'; see “Atom ID Codes” (IV–2649).

`data`

The data reference type, which is similar to the `dataRefType` parameter of `GetDataHandler` (I–407).

Parent Atom

'img' (IV–2547)

Parent atom can contain only one atom of this type.

Discussion

You use the this atom type to specify that an image is referenced and how to access it. Add this atom as a child of the 'img' (IV–2547) atom. Its ID should be 1.

' in'

Version Notes

Added in QuickTime 4.

See Also

See the `GetDataHandler` (I–407) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

' in'

Track input atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

`atomType`

Value ' in'; the first two characters are spaces.

Parent Atom

'imap' (IV–2548)

Parent atom can contain only one atom of this type.

Required Child Atoms

' ty' (IV–2606)

Input atom type. Only one allowed.

Optional Child Atoms

'obid' (IV–2571)

Object ID atom. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'kmat'

Defines a matte for a track's compressed media.

```
struct MatteCompressedAtom {
    long      size;
    long      atomType;
    long      flags;
    ImageDescription  matteImageDescription;
    char      matteData[1];
}
```


};

size

The size in bytes of this atom structure.

atomType

Constant MatteCompAID, designating atom type 'kmat'; see “Atom ID Codes” (IV–2649).

flags

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

matteImageDescription

An ImageDescription (IV–2285) data structure for the matte.

matteData

An array of matte data.

Programming Info

MoviesFormat.h

See Also

For the structure that contains this atom, see the 'matt' (IV–2557) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'krok'

Contains a KaraokeAtom (IV–2298) structure.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType

Constant 'krok'.

data

A KaraokeAtom (IV–2298) structure.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'link'

'link'

Contains specific information about a link hot spot.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `kQTVRLinkInfoAtomType`, designating atom type 'link'; see “Atom ID Codes” (IV–2649).

data

Link hot spot information.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'load'

Contains preloading information for a track.

```
struct TrackLoadSettingsAtom {  
    long          size;  
    long          atomType;  
    TrackLoadSettings settings;  
};
```

size

The size in bytes of this atom structure.

atomType

Constant `LoadSettingsAID`, designating atom type 'load'; see “Atom ID Codes” (IV–2649).

settings

A `TrackLoadSettings` (IV–2491) data structure, which contains the actual data for this atom.

Programming Info

`MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'LOOP'

User data list entry atom: looping style.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

size

The size in bytes of this atom structure.

udType

Value is 'LOOP'.

data

A long integer: 0 for none, 1 for looping, 2 for palindromic looping.

Parent Atom

'udta' (IV-2607)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

See the `MoviesUserData` (IV-2319) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'matt'

Defines a matte for a track's media.

```
struct MatteAtom {
    long                size;
    long                atomType;
    MatteCompressedAtom aCompressedMatte;
};
```

size

The size in bytes of this atom structure.

atomType

Constant `MatteAID`, designating atom type 'matt'; see “Atom ID Codes” (IV-2649).

'maxr'

aCompressedMatte

A 'kmat' (IV-2554) atom.

Required Child Atoms

'kmat' (IV-2554)

Only one allowed.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'maxr'

Maximum data rate.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II-1183) using the following parameters:

atomType

Value is 'maxr'.

data

8 bytes.

Parent Atom

'hinf' (IV-2541)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'mdat'

Media data atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II-1183) using the following parameters:

atomType

Constant MovieDataAtomType, designating atom type 'mdat'; see “Atom ID Codes” (IV–2649).

data

Media data.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'mdhd'

Specifies the characteristics of a media.

```
struct MediaHeaderAtom {  
    long        size;  
    long        atomType;  
    MediaHeader header;  
};
```

size

The size in bytes of this atom structure.

atomType

Constant MediaHeaderAID, designating atom type 'mdhd'; see “Atom ID Codes” (IV–2649).

header

A MediaHeader (IV–2305) data structure, which contains the actual data for this atom.

Parent Atom

'mdia' (IV–2560)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'mdia'

'mdia'

Defines the media for a movie track.

```
struct MediaDirectory {  
    long          size;  
    long          atomType;  
    MediaHeaderAtom  mediaHeader;  
    HandlerAtom     mediaHandler;  
    MediaInfo       mediaInfo;  
};
```

size

The size in bytes of this atom structure.

atomType

Constant MediaAID, designating atom type 'mdia'; see “Atom ID Codes” (IV–2649).

mediaHeader

A 'mdhd' (IV–2559) atom that specifies general characteristics of the media.

mediaHandler

A 'hdlr' (IV–2540) atom that defines a handler for the media.

mediaInfo

A 'minf'[generic] (IV–2562) atom structure that contains data to be passed to the media handler.

Parent Atom

'trak' (IV–2600)

Parent atom can contain only one atom of this type.

Required Child Atoms

'mdhd' (IV–2559)

General characteristics of the media. Only one allowed.

Optional Child Atoms

'hdlr' (IV–2540)

The type of media this atom contains. Only one allowed.

'minf'[generic] (IV–2562)

Data that is specific to a media handler. Only one allowed.

'udta' (IV–2607)

User data atom. Only one allowed.

Discussion

The 'hdlr' atom specifies what type of media this atom contains; for example, video or sound. The content of the 'minf'[generic] atom is specific to the media handler that is to interpret the media.

Programming Info

MoviesFormat.h

See Also

See the `MediaHeader` (IV–2305) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'mime'

Defines a graphics export MIME type.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `kGraphicsExportMIMETYPE`, designating atom type 'mime'; see “Atom ID Codes” (IV–2649).

data

A nonterminated string containing a MIME type.

Parent Atom

'expo' (IV–2535)

Parent atom can contain only one atom of this type.

See Also

See the `GraphicsImportGetExportImageTypeList` (I–632), `GraphicsImportGetMIMETYPEList` (I–641), and `GraphicsExportGetMIMETYPEList` (I–577) functions. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'minf'[base]

Provides data that is specific to a handler for media other than video or sound.

```
struct MediaInfo {
    long    size;
    long    atomType;
};
```

'minf'[generic]

size

The size in bytes of this atom structure.

atomType

Constant MediaInfoAID, designating atom type 'minf'; see “Atom ID Codes” (IV–2649).

Parent Atom

'mdia' (IV–2560)

Parent atom can contain only one atom of this type.

Required Child Atoms

'gmhd' (IV–2539)

Generic media information atom. Only one allowed.

'gmin' (IV–2540)

Provides data that is specific to a handler for media other than video or sound. Only one allowed.

Discussion

Media information atoms store handler-specific information for the media data that constitutes a track. The media handler uses this information to map from media time to media data. The format and content of media information atoms are dictated by the media handler that is responsible for interpreting the media data stream. Another media handler would not know how to interpret this information.

Programming Info

MoviesFormat.h

See Also

This isotope of the 'minf' atom provides data that is specific to a handler for media other than video or sound. Handler-specific data for sound and video are provided by the 'minf'[sound] (IV–2564) atom and the 'minf'[video] (IV–2565) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'minf'[generic]

Provides data that is specific to a handler for media other than video or sound.

```
struct MediaInfo {  
    long    size;  
    long    atomType;  
};
```


size

The size in bytes of this atom structure.

atomType

Constant MediaInfoAID, designating atom type 'minf'; see “Atom ID Codes” (IV–2649).

Parent Atom

'mdia' (IV–2560)

Parent atom can contain only one atom of this type.

Required Child Atoms

'gmhd' (IV–2539)

Generic media information atom. Only one allowed.

'hdlr' (IV–2540)

The type of media this atom contains. Only one allowed.

Optional Child Atoms

'dinf' (IV–2530)

Specifies where media data is stored. Only one allowed.

'stbl' (IV–2587)

Contains information for converting from media time to sample number to sample location and indicates how to interpret samples and chunks. Only one allowed.

Discussion

Media information atoms store handler-specific information for the media data that constitutes a track. The media handler uses this information to map from media time to media data. The format and content of media information atoms are dictated by the media handler that is responsible for interpreting the media data stream. Another media handler would not know how to interpret this information.

Programming Info

`MoviesFormat.h`

See Also

This isotope of the 'minf' atom provides data that is specific to a handler for media other than video or sound. Handler-specific data for sound and video are provided by the 'minf'[sound] (IV–2564) atom and the 'minf'[video] (IV–2565) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'minf'[sound]

'minf'[sound]

Sound media information atom.

```
struct MediaInfo {  
    long    size;  
    long    atomType;  
};
```

size

The size in bytes of this atom structure.

atomType

Constant MediaInfoAID, designating atom type 'minf'; see “Atom ID Codes” (IV-2649).

Parent Atom

'mdia' (IV-2560)

Parent atom can contain only one atom of this type.

Required Child Atoms

'smhd' (IV-2584)

Contains sound stereo balance information. Only one allowed.

'hdlr' (IV-2540)

The type of media this atom contains. Only one allowed.

Optional Child Atoms

'dinf' (IV-2530)

Specifies where media data is stored. Only one allowed.

'stbl' (IV-2587)

Contains information for converting from media time to sample number to sample location and indicates how to interpret samples and chunks. Only one allowed.

Discussion

Media information atoms store handler-specific information for the media data that constitutes a track. The media handler uses this information to map from media time to media data. The format and content of media information atoms are dictated by the media handler that is responsible for interpreting the media data stream. Another media handler would not know how to interpret this information.

Programming Info

MoviesFormat.h

See Also

This isotope of the 'minf' atom provides handler-specific data for sound. Handler-specific data for video is provided by the the 'minf'[video] (IV-2565) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'minf'[video]

Video media information atom.

```
struct MediaInfo {  
    long    size;  
    long    atomType;  
};
```

size

The size in bytes of this atom structure.

atomType

Constant MediaInfoAID, designating atom type 'minf'; see “Atom ID Codes” (IV-2649).

Parent Atom

'mdia' (IV-2560)

Parent atom can contain only one atom of this type.

Required Child Atoms

'vmhd'[media] (IV-2611)

Stores handler-specific information for video media in a track. Only one allowed.

'hdlr' (IV-2540)

The type of media this atom contains. Only one allowed.

Optional Child Atoms

'dinf' (IV-2530)

Specifies where media data is stored. Only one allowed.

'stbl' (IV-2587)

Contains information for converting from media time to sample number to sample location and indicates how to interpret samples and chunks. Only one allowed.

Discussion

Media information atoms store handler-specific information for the media data that constitutes a track. The media handler uses this information to map from media

'moov'

time to media data. The format and content of media information atoms are dictated by the media handler that is responsible for interpreting the media data stream. Another media handler would not know how to interpret this information.

Programming Info

`MoviesFormat.h`

See Also

This isotope of the 'minf' atom provides handler-specific data for video. Handler-specific data for sound is provided by the 'minf'[sound] (IV–2564) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'moov'

Contains the top-level atoms that constitute a movie.

```
struct MovieDirectory {  
    long                size;  
    long                atomType;  
    MovieHeaderAtom     header;  
    ClippingAtom        movieClip;  
    TrackDirectoryEntry track[1];  
    UserDataAtom        userData;  
};
```

size

The size in bytes of this atom structure.

atomType

Constant MovieAID, designating atom type 'moov'; see “Atom ID Codes” (IV–2649).

header

A 'mvhd' (IV–2567) atom that specifies the general characteristics of the movie.

movieClip

A 'clip' (IV–2513) atom that defines the clipping region for the movie.

track

An array of one or more TrackDirectoryEntry (IV–2489) data structures, each of which includes a 'trak' (IV–2600) atom that defines a track in the movie.

userData

A 'udta' (IV–2607) atom, which contains user data.

Required Child Atoms

'mvhd' (IV-2567)

Specifies the general characteristics of a movie. Only one allowed.

Optional Child Atoms

'clip' (IV-2513)

Defines the clipping region for the movie. Only one allowed.

'trak' (IV-2600)

Defines a single track of the movie. Any number allowed.

'udta' (IV-2607)

User data atom. Only one allowed.

'ctab' (IV-2525)

Color table atom. Only one allowed.

Discussion

You use movie atoms to specify the information that defines a movie; that is, the information that allows your application to understand the data that is stored in the movie data atom. The movie atom contains the movie header atom, which defines the time scale and duration information for the entire movie, as well as its display characteristics. In addition, the movie atom contains each track in the movie.

Programming Info

MoviesFormat.h

See Also

See the `MovieHeader` (IV-2316) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'mvhd'

Specifies the characteristics of an entire movie.

```
struct MovieHeaderAtom {  
    long        size;  
    long        atomType;  
    MovieHeader header;  
};
```

size

The size in bytes of this atom structure.

'name'[sprite]

atomType

Constant MovieHeaderAID, designating atom type 'mvhd'; see “Atom ID Codes” (IV–2649).

header

A MovieHeader (IV–2316) data structure, which contains the actual data for this atom.

Parent Atom

'moov' (IV–2566)

Parent atom can contain only one atom of this type.

Discussion

You use the movie header atom to specify the characteristics of an entire QuickTime movie. The data contained in this atom defines characteristics of the entire QuickTime movie, such as time scale and duration.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'name'[sprite]

A **sprite** name atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType

Constant kSpriteNameAtomType, designating atom type 'name'; see “Atom ID Codes” (IV–2649).

data

One or more ASCII characters comprising the **sprite**’s name.

Parent Atom

'sprt' (IV–2584)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'name'[userdata]

User data list entry atom: name of object.

```
struct MoviesUserData {  
    long    size;  
    long    udType;  
    char    data[1];  
};
```

size

The size in bytes of this atom structure.

udType

Constant kUserDataName, designating atom type 'name'; see “Atom ID Codes” (IV–2649).

data

A name string.

Parent Atom

'udta' (IV–2607)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

See the `MoviesUserData` (IV–2319) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'ndhd'

Node header atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant kQTVRNodeHeaderAtomType, designating atom type 'ndhd'; see “Atom ID Codes” (IV–2649).

data

Node header information.

'nloc'

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'nloc'

QTVR node location atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `kQTVRNodeLocationAtomType`, designating atom type 'nloc'; see “Atom ID Codes” (IV–2649).

data

Node location.

Parent Atom

'vrni' (IV–2613)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'nump'

Total number of network packets that will be sent.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Value is 'nump'.

data

8 bytes.

Parent Atom

'hinf' (IV–2541)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'obid'

Object ID atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `kTrackModifierObjectID`, designating atom type 'obid'; see “Atom ID Codes” (IV–2649).

`data`

The object ID.

Parent Atom

' in' (IV–2554)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'payt'

Payload type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Value is 'payt'.

`data`

Payload type, which includes payload number (32-bits) followed by an RTP map payload string (a Pascal string).

Parent Atom

'hinf' (IV–2541)

Parent atom can contain only one atom of this type.

'pdat'

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'pdat'

Panorama sample atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `kQTVRPanoSampleDataAtomType`, designating atom type 'pdat'; see “Atom ID Codes” (IV–2649).

data

A panorama sample.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'pmax'

Largest packet, in bytes; includes 12-byte RTP header.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Value is 'pmax'.

data

4 bytes.

Parent Atom

'hinf' (IV–2541)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'pnot'

Reference to movie **preview** data.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType

Constant ShowFilePreviewComponentType, designating atom type 'pnot'; see “Atom ID Codes” (IV–2649).

data

Reference to a movie **preview**.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'ptv '

Displays a movie in full screen mode.

This is a QT container atom; it is not declared in the header files. You can create it with QTNewAtomContainer (II–1203) and insert it with QTInsertChild (II–1183), using the following parameter:

size

Value is 0x0000000C.

atomType

Value is 'ptv '.

Parent Atom

'moov' (IV–2566)

Contains the top-level atoms that constitute a movie.

Data offsets

0x00000000

Display size: a 16-bit **little-endian** integer (see below) indicating the display size for the movie.

0x00000002

Reserved: set to 0.

'qdrq'

0x00000004

Reserved: set to 0.

0x00000006

Slide show: an 8-bit Boolean whose value is 1 for a slide show. In slide show mode, the movie advances one frame each time the right-arrow key is pressed. Audio is muted.

0x00000007

Play on open: an 8-bit boolean whose value is normally 1, indicating that the movie should play when opened. Since there is no visible controller in full-screen mode, applications should always set this field to 1 to prevent user confusion.

Display size constants

0x00000000

The movie should be played at its normal size.

0x00000001

The movie should be played at double size.

0x00000002

The movie should be played at half size.

0x00000003

The movie should be scaled to fill the screen.

0x00000004

The movie should be played at its current size. This value is normally used when the 'ptv ' atom is inserted transiently and the movie has been temporarily resized.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'qdrq'

QuickDraw region atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

atomType

Constant kTweenRegionData, designating atom type 'qdrq'; see “Atom ID Codes” (IV–2649).

data

Two Rect (IV–2399) structures and a MacRegion (IV–2303) structure.

Discussion

This atom’s ID must be 1.

See Also

See the `TweenerInitialize` (III–1977) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'rdrf'

Provides a reference to an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `ReferenceMovieDataRefAID`, designating atom type 'rdrf'; see “Atom ID Codes” (IV–2649).

data

A `ReferenceMovieDataRefRecord` (IV–2400) data structure. The alternate movie referenced by this structure is the movie associated with the parent 'rmda' (IV–2577) atom.

Parent Atom

'rmda' (IV–2577)

Parent atom can contain only one atom of this type.

Discussion

Alias data references are the contents of `AliasHandle` (`api>AliasHandle–AliasHandle`) structures. The QuickTime plug-in is smart enough to convert a relative **alias** to a relative URL. To designate the anchor file for a relative alias, pass the `FSSpec` (IV–2262) structure that specifies the file you are creating. You can pass absolute or relative URLs; if the movie is loaded from the desktop, QuickTime will convert a relative URL into a relative alias.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'reso'

'reso'

Pixmap resolution atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `kQTResolutionSettings`, designating atom type 'reso'; see “Atom ID Codes” (IV–2649).

`data`

A `QTResolutionSettings` (IV–2362) structure.

Parent Atom

'vide' (IV–2610)

Parent atom can contain only one atom of this type.

Discussion

This atom specifies the resolution for the `PixelFormat` (IV–2332) structure passed to the compressor.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'rmcd'

Provides component availability information for selecting an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `ReferenceMovieComponentCheckAID`, designating atom type 'rmcd'; see “Atom ID Codes” (IV–2649).

`data`

A `QTAltComponentCheckRecord` (IV–2345) data structure.

Parent Atom

'rmda' (IV–2577)

Parent atom can contain any number of atoms of this type.

See Also

See the `QTAltComponentCheckRecord` (IV–2345) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'rmcs'

Provides CPU speed information for selecting an alternate movie.

This is a QT atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `ReferenceMovieCPURatingAID`, designating atom type 'rmcs'; see “Atom ID Codes” (IV–2649).

`data`

A `QTAltCPURatingRecord` (IV–2346) data structure.

Parent Atom

'rmda' (IV–2577)

Parent atom can contain only one atom of this type.

See Also

See the `QTAltCPURatingRecord` (IV–2346) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'rmda'

Provides criteria for selecting an alternate movie.

This is a QT atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameter:

`atomType`

Constant `ReferenceMovieDescriptorAID`, designating atom type 'rmda'; see “Atom ID Codes” (IV–2649).

Parent Atom

'rmra' (IV–2580)

Parent atom can contain only one atom of this type.

'rmdr'

Required Child Atoms

'rdrf' (IV-2575)

A reference to an alternate movie. Only one allowed.

Optional Child Atoms

'rmdr' (IV-2578)

Data rate information for selecting an alternate movie. Only one allowed.

'rmvc' (IV-2581)

Version criteria for selecting an alternate movie. Multiples allowed.

'rmcd' (IV-2576)

Component availability information for selecting an alternate movie. Multiples allowed.

'rmqu' (IV-2579)

Playback quality information for selecting an alternate movie. Only one allowed.

'rmla' (IV-2579)

Language information for selecting an alternate movie. Only one allowed.

'rmcs' (IV-2577)

CPU speed information for selecting an alternate movie. Only one allowed.

Discussion

The 'rdrf' atom contains a `ReferenceMovieDataRefRecord`, which designates an alternate movie. The 'rmda' atom's optional atoms help QuickTime decide whether or not to run that movie. If multiple 'rmvc' or 'rmcd' atoms are present, all their criteria must be satisfied for the movie to play.

See Also

See the `ReferenceMovieDataRefRecord` (IV-2400) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'rmdr'

Provides data rate information for selecting an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

`atomType`

Constant `ReferenceMovieDataRateAID`, designating atom type 'rmdr'; see “Atom ID Codes” (IV-2649).

data

A QTAltDataRateRecord (IV–2347) data structure.

Parent Atom

'rmda' (IV–2577)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'rmla'

Provides language information for selecting an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType

Constant ReferenceMovieLanguageAID, designating atom type 'rmla'; see “Atom ID Codes” (IV–2649).

data

A QTAltLanguageRecord (IV–2348) structure.

Parent Atom

'rmda' (IV–2577)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'rmqu'

Provides playback quality information for selecting an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType

Constant ReferenceMovieQualityAID, designating atom type 'rmqu'; see “Atom ID Codes” (IV–2649).

'rmra'

data

A quality value of type SInt32. Higher quality values are selected over lower quality values.

Parent Atom

'rmda' (IV-2577)

Parent atom can contain only one atom of this type.

Discussion

If the criteria established by the 'rmdr' (IV-2578), 'rmvc' (IV-2581), and 'rmdc' (IV-2576) atoms are equally satisfied by two or more alternate movies, the one with the highest quality value will be selected.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'rmra'

Designates a reference movie.

This is a QT atom; it is not declared in the header files. You can create it with QTInsertChild (II-1183) using the following parameter:

atomType

Constant ReferenceMovieRecordAID, designating atom type 'rmra'; see “Atom ID Codes” (IV-2649).

Parent Atom

'moov' (IV-2566)

Parent atom can contain only one atom of this type.

Required Child Atoms

'rmda' (IV-2577)

Provides criteria for selecting an alternate movie. Only one allowed.

Discussion

You insert an 'rmra' atom in a 'moov' atom to create a reference movie. Each 'rmda' atom in the 'rmra' atom designates an alternate movie.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'rmvc'

Provides version criteria for selecting an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType

Constant ReferenceMovieVersionCheckAID, designating atom type 'rmvc'; see “Atom ID Codes” (IV–2649).

data

A QTAltVersionCheckRecord (IV–2348) data structure.

Parent Atom

'rmda' (IV–2577)

Parent atom can contain any number of atoms of this type.

Discussion

This optional atom in a 'rmda' atom lets you demand minimum product version criteria for selecting an alternate movie. For example, a movie that needs QuickTime VR 2.1 or later could require a Gestalt 'qtvv' value of 0x02100000 or higher.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'scpt'

Transcript track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType

Value is 'scpt'.

data

A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

'tref' (IV–2602)

Parent atom can contain only one atom of this type.

'sdp '

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'sdp '

SDP text for a hint track.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Value is 'sdp '. The fourth character is a space.

`data`

SDP text.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'sean'

The outermost atom container, of which all other atoms are children.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203), using the following parameter:

`atomData`

A pointer to a memory location that will hold the new atom.

Discussion

After creating a 'sean' atom, you can populate it with other atoms by using `QTInsertChild` (II–1183).

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'SelO'

User data list entry atom: play selection only.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

size

The size in bytes of this atom structure.

udType

Constant 'Se10'.

data

A byte indicating that only the selected area of the movie should be played.

Parent Atom

'udta' (IV-2607)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

See the `MoviesUserData` (IV-2319) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'skip'

Unused space available in the file.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

atomType

Constant `SkipAtomType`, designating atom type 'skip'; see “Atom ID Codes” (IV-2649).

data

Any number of bytes of free space.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'smhd'

'smhd'

Contains sound stereo balance information.

```
struct SoundMediaInfoHeaderAtom {  
    long          size;  
    long          atomType;  
    SoundMediaInfoHeader smiHeader;  
};
```

size

The size in bytes of this atom structure.

atomType

Constant `SoundMediaInfoHeaderAID`, designating atom type 'smhd'; see “Atom ID Codes” (IV–2649).

smiHeader

A `SoundMediaInfoHeader` (IV–2453) data structure, which contains the actual data for this atom.

Discussion

The `SoundMediaInfoHeader` data structure currently contains only stereo balance information.

Programming Info

`MoviesFormat.h`

See Also

For the structure that contains this atom, see 'minf'[sound] (IV–2564). For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'sprt'

A **key frame sprite** definition.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `kSpriteAtomType`, designating atom type 'sprt'; see “Atom ID Codes” (IV–2649).

data

A list of **sprite** property constants (see below).

Sprite property constants

kSpritePropertyMatrix

(Value is 1). Describes the **sprite**'s location and scaling within its sprite world or sprite track. By modifying a sprite's matrix, you can modify the sprite's location so that it appears to move in a smooth path on the screen or so that it jumps from one place to another. You can modify a sprite's size, so that it shrinks, grows, or stretches. Depending on which image compressor is used to create the sprite images, other transformations, such as rotation, may be supported as well. Translation-only matrixes provide the best performance.

kSpritePropertyVisible

(Value is 4). Specifies whether or not the **sprite** is visible. To make a sprite visible, you set the sprite's visible property to true.

kSpritePropertyLayer

(Value is 5). Contains a 16-bit integer value specifying the layer into which the **sprite** is to be drawn. Sprites with lower layer numbers appear in front of sprites with higher layer numbers. To designate a sprite as a background sprite, you should assign it the special layer number kBackgroundSpriteLayerNum.

kSpritePropertyGraphicsMode

(Value is 6). Specifies a graphics mode and blend color that indicates how to blend a **sprite** with any sprites behind it and with the background. To set a sprite's graphics mode, you call SetSpriteProperty (III-1641), passing a pointer to a ModifierTrackGraphicsModeRecord (IV-2311) structure.

kSpritePropertyActionHandlingSpriteID

(Value is 8). Specifies another **sprite** by ID that delegates QT events.

kSpritePropertyImageIndex

(Value is 100). Contains the atom ID of the **sprite**'s image atom.

kSpriteUsesImageIDsAtomType

(Value is 'uses'). Lets a **sprite** specify the subset of images that kSpritePropertyImageIndex can refer to.

Parent Atom

'stss' (IV-2593)

Parent atom can contain only one atom of this type.

Discussion

Sprite atoms should have ID numbers start at 1 and count consecutively upward.

'sptl'

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'sptl'

Specifies which graphics export compressor to use, its depth, and the spatial quality.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

size

The size in bytes of this atom structure.

atomType

Constant `scSpatialSettingsType`, designating atom type 'sptl'; see “Atom ID Codes” (IV–2649).

data

A pointer to `SCSpatialSettings` (IV–2423) structure.

See Also

See the `GraphicsExportCanUseCompressor` (I–553) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'ssrc'

Nonprimary source track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `kTrackModifierReference`, designating atom type 'ssrc'; see “Atom ID Codes” (IV–2649).

data

A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

'tref' (IV-2602)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'sstr'

Specifies the ID of a string variable, contained in a **sprite** track, to display in the status area of the browser.

This is a QT container atom; it is not declared in the header files. You can create it with QTNewAtomContainer (II-1203) and insert it with QTInsertChild (II-1183), using the following parameter:

atomType

Value is 'sstr'.

Parent Atom

'beha' (IV-2512)

Defines **sprite** behavior.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'stbl'

Contains information for converting from media time to sample number to sample location and indicates how to interpret samples and chunks.

```
struct SampleTableAtom {
    long                size;
    long                atomType;
    SampleDescriptionAtom sampleDescription;
    TimeToSampleNumAtom timeToSampleNum;
    SampleToChunkAtom   sampleToChunk;
    SyncSampleAtom      syncSample;
    SampleSizeAtom      sampleSize;
    ChunkOffsetAtom     chunkOffset;
    ShadowSyncAtom      shadowSync;
};
```

'stbl'

size

The size in bytes of this atom structure.

atomType

Constant SampleTableAID, designating atom type 'stbl'; see “Atom ID Codes” (IV–2649).

sampleDescription

A 'std' (IV–2591) atom, which contains information required to decode the samples in the media.

timeToSampleNum

A 'stts' (IV–2595) atom that relates sample numbers to sample durations.

sampleToChunk

A 'stsc' (IV–2590) atom that maps sample numbers to chunk numbers.

syncSample

A 'stss' (IV–2593) atom that identifies the **key frames** in the media.

sampleSize

A 'stsz' (IV–2594) atom, which identifies the size of each sample in the media.

chunkOffset

A 'stco' (IV–2589) atom that identifies the location of each data chunk in the media.

shadowSync

A 'stsh' (IV–2592) atom, which lists self-contained **sync samples** that are alternates for existing frame difference samples. This field may be omitted.

Discussion

This atom contains the information you need to find a sample number based on a time and to find the sample's location based on the sample number. Samples are organized into chunks, containing one or more samples. This atom contains the information you need to find out which chunk holds a given sample, where that chunk begins, and where in that chunk you can find the sample.

Programming Info

MoviesFormat.h

See Also

For the atoms that may contain this atom, see 'minf'[generic] (IV–2562), 'minf'[sound] (IV–2564), and 'minf'[video] (IV–2565). For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'stco'

Identifies the location of each chunk of data in the media's data stream.

```
struct ChunkOffsetAtom {
    long    size;
    long    atomType;
    long    flags;
    long    numEntries;
    long    chunkOffsetTable[1];
};
```

size

The size in bytes of this atom structure.

atomType

Constant STChunkOffsetAID, designating atom type 'stco'; see “Atom ID Codes” (IV–2649).

flags

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

numEntries

The number of entries in chunkOffsetTable.

chunkOffsetTable

An array of chunk offset values.

Discussion

A chunk is a collection of data samples in a media that allows optimized data access. A chunk may contain one or more samples. Chunks in a media may have different sizes, and the samples within a chunk may have different sizes.

Programming Info

`MoviesFormat.h`

See Also

For the structure that contains this atom, see 'stbl' (IV–2587). For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'strt'

Defines the starting offset of hypertext in a text stream.

'strv'

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Value is 'strt'.

data

The starting offset of hypertext.

Parent Atom

'htxt' (IV–2546)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'strv'

Contains a string variable for a **sprite**.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

atomType

Value is 'strv'.

Parent Atom

'vars' (IV–2610)

Contains variables for a **sprite**.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'stsc'

Maps sample numbers to chunk numbers.

```
struct SampleToChunkAtom {
    long          size;
    long          atomType;
    long          flags;
```

```

        long            numEntries;
        SampleToChunk   sampleToChunkTable[1];
};

```

size

The size in bytes of this atom structure.

atomType

Constant STSampleToChunkAID, designating atom type 'stsc'; see “Atom ID Codes” (IV–2649).

flags

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

numEntries

The number of entries in sampleToChunkTable.

sampleToChunkTable

An array of SampleToChunk (IV–2417) data structures, which contain the actual data for this atom.

Discussion

A chunk is a collection of data samples in a media that allows optimized data access. A chunk may contain one or more samples. Chunks in a media may have different sizes, and the samples within a chunk may have different sizes.

Programming Info

MoviesFormat.h

See Also

For the structure that contains this atom, see 'stbl' (IV–2587). For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'std'

Holds one or more sample description structures.

```

struct SampleDescriptionAtom {
    long            size;
    long            atomType;
    long            flags;
    long            numEntries;
    SampleDescription   sampleDescTable[1];
};

```

'stsh'

size

The size in bytes of this atom structure.

atomType

Constant STSampleDescAID, designating atom type 'stsd'; see “Atom ID Codes” (IV–2649).

flags

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

numEntries

Number of entries in sampleDescTable.

sampleDescTable

An array of SampleDescription (IV–2414) data structures, which contain the actual data for this atom.

Discussion

In QuickTime streaming, this atom describes the data format of the hint samples and contains track-level information, such as the RTP timescale for a track.

Programming Info

MoviesFormat.h

See Also

For the structure that contains this atom, see 'stbl' (IV–2587). For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'stsh'

Lists self-contained **sync samples** that are alternates for existing frame difference samples.

```
struct ShadowSyncAtom {
    long        size;
    long        atomType;
    long        flags;
    long        numEntries;
    ShadowSync  shadowSyncTable[1];
};
```

size

The size in bytes of this atom structure.

atomType

Constant STShadowSyncAID, designating atom type 'stsh'; see “Atom ID Codes” (IV–2649).

flags

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

numEntries

The number of entries in shadowSyncTable.

shadowSyncTable

An array of ShadowSync (IV–2436) data structures, which contain the actual data for this atom.

Discussion

Shadow sync atoms are used to optimize random access operations on a movie, thereby enhancing playback performance.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'stss'

Identifies the **key frames** in a media.

```
struct SyncSampleAtom {
    long    size;
    long    atomType;
    long    flags;
    long    numEntries;
    long    syncSampleTable[1];
};
```

size

The size in bytes of this atom structure.

atomType

Constant STSyncSampleAID, designating atom type 'stss'; see “Atom ID Codes” (IV–2649).

flags

Flags (currently not used).

'stsz'

numEntries

The number of entries in syncSampleTable.

syncSampleTable

An array of physical sample numbers, each of which identifies a **key frame** in the media.

Discussion

In a media that contains compressed data, **key frames** define starting points for portions of a temporally compressed sequence. Each key frame is independent of the preceding frames. Subsequent frames may depend on the key frame.

Programming Info

MoviesFormat.h

See Also

For the structure that contains this atom, see 'stbl' (IV-2587). For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'stsz'

Identifies the size of each sample in a media.

```
struct SampleSizeAtom {  
    long    size;  
    long    atomType;  
    long    flags;  
    long    sampleSize;  
    long    numEntries;  
    long    sampleSizeTable[1];  
};
```

size

The size in bytes of this atom structure.

atomType

Constant STSampleSizeAID, designating atom type 'stsz'; see “Atom ID Codes” (IV-2649).

flags

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

sampleSize

The number of bytes in the samples. If all the samples are the same size, sampleSize indicates the size of all the samples. If sampleSize is set to 0, then the samples have different sizes, and those sizes are stored in sampleSizeTable.

numEntries

The number of entries in sampleSizeTable.

sampleSizeTable

An array of numbers, one for every sample. This field is indexed by sample number; the first entry corresponds to the first sample, the second to the second sample, and so on.

Programming Info

MoviesFormat.h

See Also

For the structure that contains this atom, see 'stbl' (IV–2587). For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'stts'

Holds one or more structures that relate sample numbers to sample durations.

```
struct TimeToSampleNumAtom {
    long          size;
    long          atomType;
    long          flags;
    long          numEntries;
    TimeToSampleNum  timeToSampleNumTable[1];
};
```

size

The size in bytes of this atom structure.

atomType

Constant STTimeToSampAID, designating atom type 'stts'; see “Atom ID Codes” (IV–2649).

flags

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

numEntries

The number of entries in timeToSampleNumTable.

'sync'

`timeToSampleNumTable`

An array of `TimeToSampleNum` (IV-2486) data structures, each of which maps a sample number to its sample duration.

Discussion

Entries in `timeToSampleNumTable` collect samples according to their order in the media and their duration. If consecutive samples have the same duration, a single table entry may be used to define more than one sample. In these cases, the `count` field indicates the number of consecutive samples that have the same duration. For example, if a video media had a constant frame rate, `timeToSampleNumTable` would have one entry.

Programming Info

`MoviesFormat.h`

See Also

For the structure that contains this atom, see `'stbl'` (IV-2587). For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'sync'

Synchronization track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

`atomType`

Value is `'sync'`.

`data`

A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

`'tref'` (IV-2602)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'tbox'

Defines a text box rectangle.

```
struct TextBoxAtom {
    long    size;
    long    atomType;
    Rect    textBox;
};
```

size

The size in bytes of this atom structure.

atomType

Value is 'tbox'.

textBox

A new text box rectangle, which overrides the rectangle defined by the defaultTextBox constant.

Discussion

This is a classic atom; you can access its information by calculating offsets.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'tcmi'

Time code media information atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II-1183) using the following parameters:

atomType

Value is 'tcmi'.

data

Time code media information.

Parent Atom

'minf'[video] (IV-2565)

Parent atom can contain only one atom of this type.

'tkhd'

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'tkhd'

Specifies the characteristics of a track in a movie.

```
struct TrackHeaderAtom {  
    long        size;  
    long        atomType;  
    TrackHeader header;  
};
```

size

The size in bytes of this atom structure.

atomType

Constant TrackHeaderAID, designating atom type 'tkhd'; see “Atom ID Codes” (IV–2649).

header

A TrackHeader (IV–2489) structure that contains the actual data for this atom.

Parent Atom

'trak' (IV–2600)

Parent atom can contain only one atom of this type.

Programming Info

MoviesFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'tmax'

Largest relative transmission time, in milliseconds.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType

Value is 'tmax'.

data
4 bytes.

Parent Atom

'hinf' (IV-2541)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'tmcd'

Time code track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

atomType
Constant `TimeCodeMediaType`, designating atom type 'tmcd'; see “Atom ID Codes” (IV-2649).

data
A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

'tref' (IV-2602)
Parent atom can contain only one atom of this type.

See Also

See the `TimeCodeDescription` (IV-2483) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'tmin'

Smallest relative transmission time, in milliseconds.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

'tpyl'

atomType
Value is 'tmin'.
data
4 bytes.

Parent Atom

'hinf' (IV-2541)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'tpyl'

Total number of bytes that will be sent, not including 12-byte RTP headers.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II-1183) using the following parameters:

atomType
Value is 'tpyl'.
data
8 bytes.

Parent Atom

'hinf' (IV-2541)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'trak'

Defines a single track of a movie.

```
struct TrackDirectory {  
    long          size;  
    long          atomType;  
    TrackHeaderAtom trackHeader;  
    ClippingAtom   trackClip;  
    EditsAtom      edits;
```

```

    MediaDirectory    media;
    UserDataAtom      userData;
};

```

size

The size in bytes of this atom structure.

atomType

Constant TrackAID, designating atom type 'trak'; see “Atom ID Codes” (IV–2649).

trackHeader

A 'tkhd' (IV–2598) atom, which specifies general characteristics of the track.

trackClip

A 'clip' (IV–2513) atom, which defines the track's clipping region.

edits

An 'edts' (IV–2533) atom, which defines the portions of the track's media that are going into the track.

media

A 'mdia' (IV–2560) atom structure that defines the media for the track.

userData

A 'udta' (IV–2607) atom that contains user data.

Parent Atom

'moov' (IV–2566)

Parent atom can contain any number of atoms of this type.

Required Child Atoms

'tkhd' (IV–2598)

Specifies general characteristics of the track. Only one allowed.

'mdia' (IV–2560)

The media for the track. Only one allowed.

Optional Child Atoms

'clip' (IV–2513)

Defines the clipping region for the track. Only one allowed.

'matt' (IV–2557)

Defines a matte for a track's media. Only one allowed.

'edts' (IV–2533)

Defines the portions of the track's media that are going into the track. Only one allowed.

'tref'

'tref' (IV-2602)

Track reference atom. Only one allowed.

'load' (IV-2556)

Contains preloading information for a track. Only one allowed.

'imap' (IV-2548)

An input map. Only one allowed.

'udta' (IV-2607)

User data atom. Only one allowed.

Discussion

A movie may consist of one or more tracks. Each track is independent of the other tracks in the movie and carries its own temporal and spatial information. Each track atom contains an associated media atom. Note that there must be at least one media track within a QuickTime movie. All media tracks that are present must remain in the movie, even if the media data within them is not referenced by the hint tracks. After deleting all hint tracks, the entire unhinted movie must remain.

Programming Info

`MoviesFormat.h`

See Also

See the `TrackDirectoryEntry` (IV-2489) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'tref'

Track reference atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II-1203) and insert it with `QTInsertChild` (II-1183), using the following parameters:

`atomType`

Constant `TrackHeaderAID`, designating atom type 'tref'; see “Atom ID Codes” (IV-2649).

`data`

One track reference atom of a type listed below.

Parent Atom

'trak' (IV-2600)

Parent atom can contain only one atom of this type.

Required Child Atoms (one from this list)`'tmcd'` (IV-2599)

Time code track reference type atom. Only one allowed.

`'chap'` (IV-2512)

Chapter or scene list track reference type atom. Only one allowed.

`'sync'` (IV-2596)

Synchronization track reference type atom. Only one allowed.

`'scpt'` (IV-2581)

Transcript track reference type atom. Only one allowed.

`'ssrc'` (IV-2586)

Nonprimary source track reference type atom. Only one allowed.

`'hint'` (IV-2542)

Hint track reference type atom. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'trpy'

Total number of bytes that will be sent, including 12-byte RTP headers, but not including network headers.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

`atomType`

Value is `'trpy'`.

`data`

8 bytes.

Parent Atom`'hinf'` (IV-2541)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'twdt'

'twdt'

Tween data type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType

Value is 'twdt'.

data

Tween data.

Parent Atom

'twen' (IV–2605)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'twdu'

Tween duration atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (II–1183) using the following parameters:

atomType

Constant kTweenDuration, designating atom type 'twdu'; see “Atom ID Codes” (IV–2649).

data

Tween duration data.

Parent Atom

'twen' (IV–2605)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'twen'

Tween entry atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

`atomType`

Constant `kTweenEntry`, designating atom type 'twen'; see “Atom ID Codes” (IV–2649).

Required Child Atoms

'twst' (IV–2606)

Tween start atom. Only one allowed.

'twdu' (IV–2604)

Tween duration atom. Only one allowed.

'twdt' (IV–2604)

Tween data type atom. Only one allowed.

'twnt' (IV–2605)

Tween type atom. Only one allowed.

See Also

See the `TweenSequenceEntryRecord` (IV–2494) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'twnt'

Tween type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`

Constant `kTweenType`, designating atom type 'twnt'; see “Atom ID Codes” (IV–2649).

`data`

Tween type.

'twst'

Parent Atom

'twen' (IV-2605)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'twst'

Tween start offset atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

atomType

Constant `kTweenStartOffset`, designating atom type 'twst'; see “Atom ID Codes” (IV-2649).

data

Tween start offset.

Parent Atom

'twen' (IV-2605)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

' ty'

Input atom type.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II-1183) using the following parameters:

atomType

Value is ' ty'; first and second characters are spaces.

data

Track input atom type code.

Parent Atom

' in' (IV-2554)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'udta'

Holds one or more structures of movie user data.

```
struct UserDataAtom {
    long          size;
    long          atomType;
    MoviesUserData userData[1];
};
```

size

The size in bytes of this atom structure.

atomType

Constant UserDataAID, designating atom type 'udta'; see “Atom ID Codes” (IV-2649).

userData

An array of MoviesUserData (IV-2319) data structures, which contain the actual data for this atom. The currently defined types are listed below.

Parent atom (one or the other)

'moov' (IV-2566)

Parent atom can contain any number of atoms of this type.

'trak' (IV-2600)

Parent atom can contain any number of atoms of this type.

Optional child atom

'@cpy' (IV-2515)

Copyright statement.

'@day' (IV-2516)

Date the movie content was created.

'@dir' (IV-2517)

Name of movie's director.

'udta'

'@ed1' (IV-2517)

First edit date and description. Similar for '@ed2' through '@ed9'.

'@fmt' (IV-2518)

Indication of movie format (computer-generated, digitized, and so on).

'@inf' (IV-2519)

Information about the movie.

'@prd' (IV-2519)

Name of movie's producer.

'@prf' (IV-2520)

Names of performers.

'@req' (IV-2521)

Special hardware and software requirements.

'@src' (IV-2521)

Credits for those who provided movie source content.

'@wrt' (IV-2522)

Name of movie's writer.

'WLOC' (IV-2615)

Default window location for movie. Two 16-bit values, {xy}.

'name'[userdata] (IV-2569)

Name of object.

'LOOP' (IV-2557)

Looping. Long integer indicating looping style. 0 for none, 1 for looping, 2 for palindromic looping.

'Se10' (IV-2582)

Play selection only. Byte indicating that only the selected area of the movie should be played.

'AllF' (IV-2511)

Play all frames. Byte indicating that all frames of video should be played, regardless of timing.

Discussion

Inside the user data atom is a list of atoms describing each piece of user data. Each data element contains size and type information along with the data. Furthermore, for historical reasons, the list of atoms is optionally terminated by a 32-bit integer set to 0. If you are writing a program to read user data atoms, you should allow for the terminating 0. However, if you are writing a program to create user data atoms, you can safely leave out the trailing 0.

Programming Info

MoviesFormat.h

See Also

See the `MoviesUserData` (IV–2319) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'url '

Contains a URL for a **sprite**.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

```
atomType
    Value is 'url '.
```

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'uses'

Lets a **sprite** specify the subset of images that its image index property can refer to.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

```
atomType
    Value is 'uses'.
```

Parent Atom

'sprt' (IV–2584)
A **key frame sprite** definition.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'vars'

'vars'

Contains variables for a **sprite**.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

`atomType`

Value is 'vars'.

Optional Child Atoms

'flv' (IV–2537)

Contains a floating-point variable for a **sprite**. Multiple atoms allowed.

'strv' (IV–2590)

Contains a string variable for a **sprite**. Multiple atoms allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'vide'

Contains compression information for the Base Image Exporter.

This is a QT atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameter:

`atomType`

Constant `VideoMediaType`, designating atom type 'vide'; see “Component Identifiers” (IV–2657).

Optional Child Atoms

'dasz' (IV–2527)

Only one allowed. If present, it specifies a desired data size. The base exporter repeats compression attempts, decreasing the `quality` parameter until it reaches this size or lower, or it runs out of patience.

'reso' (IV–2576)

Only one allowed. If present, it specifies the resolution for the `pixmap` passed to the compressor.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'vmhd'[media]

Stores handler-specific information for video media in a track.

```
struct VideoMediaInfo {
    long                size;
    long                atomType;
    VideoMediaInfoHeaderAtom header;
    HandlerAtom         dataHandler;
    DataInfoAtom        dataInfo;
    SampleTableAtom     sampleTable;
};
```

size

The size in bytes of this atom structure.

atomType

Constant VideoMediaInfoAID, designating atom type 'vmhd'; see “Atom ID Codes” (IV–2649).

header

A 'vmhd'[transfer] (IV–2612) atom, which defines the graphics transfer mode for this video media.

dataHandler

A 'hdlr' (IV–2540) atom that specifies the component that is to handle this media.

dataInfo

A 'dinf' (IV–2530) atom, which specifies where the video media data is stored.

sampleTable

A 'stbl' (IV–2587) atom, which tells the media handler how to locate and interpret data samples.

Discussion

This atom stores handler-specific information for the media that constitutes a video track. A video media handler uses this information to map from media time to media data. Another type of media handler would not know how to interpret this information.

'vmhd'[transfer]

Programming Info

MoviesFormat.h

See Also

See the 'minf'[sound] (IV–2564) atom for sound media and the 'minf'[generic] (IV–2562) atom for media other than video or sound. Also see the VideoMediaInfoHeader (IV–2503) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'vmhd'[transfer]

Defines the graphics transfer characteristics for a video media.

```
struct VideoMediaInfoHeaderAtom {  
    long          size;  
    long          atomType;  
    VideoMediaInfoHeader vmiHeader;  
};
```

size

The size in bytes of this atom structure.

atomType

Constant VideoMediaInfoHeaderAID, designating atom type 'vmhd'; see “Atom ID Codes” (IV–2649).

vmiHeader

A VideoMediaInfoHeader (IV–2503) data structure, which contains the actual data for this atom.

Programming Info

MoviesFormat.h

See Also

For the structure that contains this atom, see 'minf'[video] (IV–2565). Also see the VideoMediaInfoHeader (IV–2503) structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'vrcp'

Custom cursor atom parent.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

atomType

Constant `kQTVRCursorParentAtomType`, designating atom type 'vrnp'; see “Atom ID Codes” (IV–2649).

Required Child Atoms

'CURS' (IV–2526)

Custom cursor child atom. Multiple atoms of this type allowed.

'crsr' (IV–2524)

Color custom cursor child atom. Multiple atoms of this type allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'vrni'

QTVR node ID atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

atomType

Constant `kQTVRNodeIDAtomType`, designating atom type 'vrni'; see “Atom ID Codes” (IV–2649).

Parent Atom

'vrnp' (IV–2614)

Parent atom can contain any number of atoms of this type.

Required Child Atoms

'nloc' (IV–2570)

QTVR node location atom. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'vrnp'

'vrnp'

QTVR node parent atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

atomType

Constant `kQTVRNodeParentAtomType`, designating atom type 'vrnp'; see “Atom ID Codes” (IV–2649).

Required Child Atoms

'vrni' (IV–2613)

QTVR node ID atom. Any number allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'vrsc'

VR world header atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

atomType

Constant `kQTVRWorldHeaderAtomType`, designating atom type 'vrsc'; see “Atom ID Codes” (IV–2649).

data

Contains the name of the scene and the default node ID.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'wide'

Wide atom name placeholder atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (II–1183) using the following parameters:

`atomType`
 Constant `WideAtomPlaceholderType`, designating atom type 'wide'; see “Atom ID Codes” (IV–2649).
`data`
 8 bytes of placeholder space to allow an atom to be converted from a 32-bit to a 64-bit atom.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'WLOC'

User data list entry atom: default window location for movie.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

`size`
 The size in bytes of this atom structure.

`udType`
 Value is 'WLOC'.

`data`
 2 16-bit values, {xy}.

Parent Atom

'udta' (IV–2607)
 Parent atom can contain only one atom of this type.

Programming Info

`MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'wtxt'

'wtxt'

Parent atom for hypertext items.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (II–1203) and insert it with `QTInsertChild` (II–1183), using the following parameter:

```
atomType
    Value is 'wtxt'.
```

Required Child Atoms

```
'strt' (IV-2589)
    Defines the starting offset of hypertext in a text stream. Only one allowed.
'end ' (IV-2534)
    Defines the ending offset of hypertext in a text stream. Only one allowed.
```

Optional Child Atoms

```
'htxt' (IV-2546)
    Multiple atoms allowed.
```

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

'vrsg'

Contains the name of a VR scene.

This is a QT leaf atom that contains a struct in its data field. Using the constant `kQTVRStringAtomType` and a pointer to the atom, you can access its data with `QTGetAtomDataPtr` (II–1171) and change it with `QTSetAtomData` (II–1223). The struct is declared as follows:

```
struct QTVRStringAtom {
    UInt16      stringUsage;
    UInt16      stringLength;
    unsigned char theString[4];
};
```

```
stringUsage
    Unused field.
```

stringLength

The length of the name string in bytes.

theString

The name as a C string.

Discussion

One leaf atom of this type is contained in a VR world container. You can get a pointer to this container by calling `QTVRGetVRWorld` (II–1385). One of this atom's siblings in the VR world is a 'vrsc' (IV–2614) atom, which contains the atom ID of this atom in its `nameAtomID` field. The following code illustrates a function that returns the name of a VR node as a Pascal string, given the node's ID:

```
OSErr MyGetNodeName (QTVRInstance theInstance, UInt32 theNodeID,
                    StringPtr theStringPtr)
{
    OSErr                theErr = noErr;
    QTAtomContainer      theNodeInfo;
    QTVRNodeHeaderAtomPtr theNodeHeader;
    QTAtom               theNodeHeaderAtom = 0;

    // Get the node information atom container
    theErr = QTVRGetNodeInfo(theInstance, theNodeID, &theNodeInfo);

    // Get the node header atom.
    if (!theErr)
        theNodeHeaderAtom = QTFindChildByID(theNodeInfo,
                                             kParentAtomIsContainer,
                                             kQTVRNodeHeaderAtomType, 1, nil);

    if (theNodeHeaderAtom != 0) {
        QTLockContainer(theNodeInfo);

        // Get a pointer to the node header atom data.
        theErr = QTGetAtomDataPtr(theNodeInfo, theNodeHeaderAtom, nil,
                                   (Ptr *)&theNodeHeader);

        // See if there is a name atom.
        if (!theErr && theNodeHeader->nameAtomID != 0) {
            QTAtom theNameAtom;
            theNameAtom = QTFindChildByID(theNodeInfo,
                                           kParentAtomIsContainer, kQTVRStringAtomType,
                                           theNodeHeader->nameAtomID, nil);

            if (theNameAtom != 0) {
                VRStringAtomPtr theStringAtomPtr;
```

'vrsg'

```
        // Get a pointer to the name atom data; copy it into string
        theErr = QTGetAtomDataPtr(theNodeInfo, theNameAtom, nil,
                                   (Ptr *)&theStringAtomPtr);

        if (!theErr) {
            short theLen = theStringAtomPtr->stringLength;
            if (theLen > 255)
                theLen = 255;
            BlockMove(theStringAtomPtr->theString,
                      &theStringPtr[1], theLen);
            theStringPtr[0] = theLen;
        }
    }
    QTUnlockContainer(theNodeInfo);
}

QTDisposeAtomContainer(theNodeInfo);
return(theErr);
}
```

Programming Info

QuickTimeVRFormat.h

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

Defined Data Types

Atom Types

Provide access to data stored in QuickTime atoms.

```
typedef Handle QTAtomContainer;
```

```
typedef long QTAtom;
```

```
typedef long QTAtomType;
```

```
typedef long QTAtomID;
```

```
typedef long DataRefAtom;
```

Special Considerations

The `DataRefAtom` type is a private structure and is not documented.

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 18.

Codec Types

Support codec components and the Image Compression Manager.

```
typedef Component CompressorComponent;
```

```
typedef Component DecompressorComponent;
```

```
typedef Component CodecComponent;
```

```
typedef OSType CodecType;
```

```
typedef unsigned short CodecFlags;
```

```
typedef unsigned long CodecQ;
```

```
typedef CDSequenceDataSourceQueueEntry * CDSequenceDataSourceQueueEntryPtr;
```

```
typedef CDSequenceDataSource * CDSequenceDataSourcePtr;
```

```

union SourceData {
    CDSequenceDataSourcePtr    image;
    EffectSourcePtr            effect;
};

typedef ICMFrameTimeInfo * ICMFrameTimeInfoPtr;

typedef ICMProgressProcRecord * ICMProgressProcRecordPtr;

typedef ICMCompletionProcRecord * ICMCompletionProcRecordPtr;

typedef ICMDDataProcRecord * ICMDDataProcRecordPtr;

typedef ICMFlushProcRecord * ICMFlushProcRecordPtr;

typedef ICMAAlignmentProcRecord * ICMAAlignmentProcRecordPtr;

typedef DataRateParams * DataRateParamsPtr;

typedef ImageDescription * ImageDescriptionPtr;

typedef ImageDescriptionPtr * ImageDescriptionHandle;

typedef CodecNameSpecList * CodecNameSpecListPtr;

typedef ICMFrameTimeRecord * ICMFrameTimePtr;

typedef PreviewResourceRecord * PreviewResourcePtr;

typedef PreviewResourcePtr * PreviewResource;

typedef ICMPixelFormatInfo * ICMPixelFormatInfoPtr;

typedef unsigned short MatrixFlags;

typedef void * ICMCursorNotify;

typedef long ImageSequence;

typedef long ImageSequenceDataSource;

typedef long ImageTranscodeSequence;

typedef long ImageFieldSequence;

```

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 5.

Component Types

Define component types and support QuickTime component interfaces.

```
typedef ComponentResource * ComponentResourcePtr;

typedef ComponentResourcePtr * ComponentResourceHandle;

typedef ExtComponentResource * ExtComponentResourcePtr;

typedef ExtComponentResourcePtr * ExtComponentResourceHandle;

typedef ComponentRecord * Component;

typedef ComponentInstanceRecord * ComponentInstance;

typedef RegisteredComponentRecord * RegisteredComponentRecordPtr;

typedef RegisteredComponentRecord * RegisteredComponentPtr;

typedef RegisteredComponentInstanceRecord *
RegisteredComponentInstanceRecordPtr;

typedef RegisteredComponentInstanceRecord * RegisteredComponentInstancePtr;

typedef long ComponentResult;

typedef UniversalProcPtr ComponentFunctionUPP;

typedef const Component * ConstComponentListPtr;

typedef ComponentInstance GraphicsImportComponent;

typedef ComponentInstance GraphicsExportComponent;

typedef ComponentInstance ImageTranscoderComponent;

typedef ComponentInstance MovieImportComponent;

typedef ComponentInstance MovieExportComponent;

typedef ComponentInstance TextExportComponent;

typedef ComponentInstance GraphicImageMovieImportComponent;

typedef ComponentInstance pnotComponent;

typedef ComponentInstance DataCompressorComponent;
```

TYP

Defined Data Types

```
typedef ComponentInstance DataDecompressorComponent;
typedef ComponentInstance DataCodecComponent;
typedef ComponentInstance SequenceFilterComponent;
typedef ComponentInstance MediaHandler;
typedef Component MediaHandlerComponent;
typedef ComponentInstance TweenerComponent;
typedef QTweenerRecord * QTweener;
```

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 9.

Data Handling Types

Support data handler components.

```
typedef ComponentInstance DataHandler;
typedef Component DataHandlerComponent;
typedef DataHVolumeListRecord * DataHVolumeListPtr;
typedef DataHVolumeListPtr * DataHVolumeList;
typedef DataHScheduleRecord * DataHSchedulePtr;
typedef ComponentResult HandlerError;
typedef OSType * DataHFileTypeOrderingPtr;
typedef DataHFileTypeOrderingPtr * DataHFileTypeOrderingHandle;
```

Effects Types

Support visual effects and the standard parameters dialog box.

```
typedef EffectSource * EffectSourcePtr;
typedef EffectsFrameParams * EffectsFrameParamsPtr;
```

```

typedef long QTParaterValidationOptions;

typedef long SMPTEFlags;

typedef long SMPTEFrameReference;

typedef unsigned long SMPTEWipeType;

typedef GradientColorRecord * GradientColorPtr;

typedef long GradientType;

typedef MatrixRecord * MatrixRecordPtr;

typedef QTParamPreviewRecord * QTParamPreviewPtr;

typedef QTParamDialogEventRecord * QTParamDialogEventPtr;

typedef QTParamFetchPreviewRecord * QTParamFetchPreviewPtr;

typedef long QTParaterDialog;

typedef long QTEffectListOptions;

typedef long QTParaterDialogOptions;

```

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 14.

Endian Types

Define various **big-endian** and **little-endian** formats for conversion functions.

```

typedef long BigEndianLong;

typedef unsigned long BigEndianUnsignedLong;

typedef short BigEndianShort;

typedef unsigned short BigEndianUnsignedShort;

typedef Fixed BigEndianFixed;

typedef UnsignedFixed BigEndianUnsignedFixed;

typedef OSType BigEndianOSType;

```

File System Types

Provide access to Macintosh file system specifications and file **aliases**.

```
typedef FSSpec * FSSpecPtr;

typedef FSSpecPtr * FSSpecHandle;

typedef FSSpecPtr FSSpecArrayPtr;

typedef const FSSpec * ConstFSSpecPtr;

typedef AliasRecord * AliasPtr;

typedef AliasPtr * AliasHandle;

typedef short AliasInfoType;

typedef OSType SFTYPEList;

typedef CInfoPBRec * CInfoPBPtr;           // in Files.h
```

See Also

See *Inside Macintosh: Files* (listed in the **bibliography**).

Graphics Types

Support Windows and Macintosh graphics worlds.

```
typedef GDevice * GDPtr;

typedef GDPtr * GDHandle;

typedef GrafPort * GrafPtr;

typedef GrafPtr WindowPtr;

typedef struct GrafVars GrafVars;         // in Quickdraw.h

typedef SInt8 GrafVerb;                   // in Quickdraw.h

typedef WindowPtr DialogPtr;

typedef WindowPtr WindowRef;

typedef CGrafPort * CGrafPtr;
```

```
typedef CGrafPtr CWindowPtr;  
typedef CGrafPtr GWorldPtr;  
typedef unsigned long GWorldFlags;  
typedef BitMap * BitMapPtr;  
typedef BitMapPtr * BitMapHandle;  
typedef Picture * PicPtr;  
typedef PicPtr * PicHandle;  
typedef PixMapExtension * PixMapExtPtr;  
typedef PixMapExtPtr * PixMapExtHandle;  
typedef PixMap * PixMapPtr;  
typedef PixMapPtr * PixMapHandle;  
typedef PixPat * PixPatPtr;  
typedef PixPatPtr * PixPatHandle;  
typedef short CharParameter;  
typedef unsigned char Style;  
typedef short StyleParameter;  
typedef Style StyleField;  
typedef RGBColor * RGBColorPtr;  
typedef RGBColorPtr * RGBColorHdl;  
typedef VGammaRecord * VDGamRecPtr;  
typedef UInt16 DragConstraint;  
typedef ColorTable * CTabPtr;  
typedef CTabPtr * CTabHandle;  
typedef MacPolygon Polygon;
```

```

typedef MacPolygon * PolyPtr;

typedef PolyPtr * PolyHandle;

typedef ColorSpec * ColorSpecPtr;

typedef ColorSpec CSpecArray;

typedef CProcRec * CProcPtr;

typedef CProcPtr * CProcHndl;

typedef CQDProcs * CQDProcsPtr;

typedef QDProcs * QDProcsPtr;

typedef ITab * ITabPtr;

typedef ITabPtr * ITabHandle;

```

See Also

See *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

Media Handling Types

Support QuickTime media handlers.

```

typedef Handle * dataHandlePtr;

typedef dataHandlePtr * dataHandleHandle;

typedef SampleDescription * SampleDescriptionPtr;

typedef SampleDescriptionPtr * SampleDescriptionHandle;

typedef SampleReferenceRecord * SampleReferencePtr;

typedef SampleReference64Record * SampleReference64Ptr;

typedef DataReferenceRecord * DataReferencePtr;

typedef QTEventRecord * QTEventRecordPtr;

typedef QTAtomSpec * QTAtomSpecPtr;

typedef ResolvedQTEventSpec * ResolvedQTEventSpecPtr;

```



```

typedef unsigned long mediaHandlerFlagsEnum;

typedef QTCustomActionTargetRecord * QTCustomActionTargetPtr;

typedef MediaEQSpectrumBandsRecord * MediaEQSpectrumBandsRecordPtr;

typedef FlashDescription * FlashDescriptionPtr;

typedef FlashDescriptionPtr * FlashDescriptionHandle;

typedef ThreeDeeDescription * ThreeDeeDescriptionPtr;

typedef ThreeDeeDescriptionPtr * ThreeDeeDescriptionHandle;

```

Memory Types

Provide access to memory.

```

typedef char * Ptr;

typedef Ptr * Handle;

typedef long Size;

typedef void * LogicalAddress;

typedef const void * ConstLogicalAddress;

typedef void * PhysicalAddress;

typedef UInt8 * BytePtr;

typedef UInt32 ByteCount;

typedef UInt32 ByteOffset;

typedef Point * PointPtr;

typedef Rect * RectPtr;

typedef MacRegion Region;

typedef MacRegion * RgnPtr;

typedef RgnPtr * RgnHandle;

typedef QElem * QElemPtr;

```

TYP

Defined Data Types

```
typedef QHdr * QHdrPtr;
```

Movie Controller Types

Support movie controller interfaces.

```
typedef ComponentInstance MovieController;  
typedef MovieController * MovieControllerPtr;  
typedef unsigned long MCInterfaceElement;  
typedef short mcAction;  
typedef unsigned long mcFlags;  
typedef QTStatusStringRecord * QTStatusStringPtr;  
typedef QTGetExternalMovieRecord * QTGetExternalMoviePtr;  
typedef QTGetChapterTimeRecord * QTGetChapterTimePtr;  
typedef QTChapterInfoRecord * QTChapterInfoPtr;  
typedef QTEvaluateExpressionRecord * QTEvaluateExpressionPtr;  
typedef QTFetchParameterAsRecord * QTFetchParameterAsPtr;  
typedef QTGetCursorByIDRecord * QTGetCursorByIDPtr;  
typedef QTDoScriptRecord * QTDoScriptPtr;  
typedef QTRestartAtTimeRecord * QTRestartAtTimePtr;
```

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 10.

Movie Types

Support movie creation and editing.

```
typedef MovieRecord * Movie;  
typedef Movie * MoviePtr;
```

```

typedef TrackRecord * Track;

typedef MediaRecord * Media;

typedef UserDataRecord * UserData;

typedef TrackEditStateRecord * TrackEditState;

typedef struct TrackDirectory TrackDirectory;    // see 'trak' atom.

typedef MovieEditStateRecord * MovieEditState;

typedef unsigned long createMovieFileFlagsEnum;

typedef unsigned long movieFlattenFlagsEnum;

typedef unsigned long dataRefAttributesFlags;

typedef unsigned long playHintsEnum;

typedef ConnectionSpeedPrefsRecord * ConnectionSpeedPrefsPtr;

typedef ConnectionSpeedPrefsPtr * ConnectionSpeedPrefsHandle;

typedef BandwidthManagementPrefsRecord * BandwidthManagementPrefsPtr;

typedef BandwidthManagementPrefsPtr * BandwidthManagementPrefsHandle;

typedef struct OpaqueQTBandwidthReference * QTBandwidthReference;

typedef struct OpaqueQTScheduledBandwidthReference *
QTScheduledBandwidthReference;

typedef QTScheduledBandwidthRecord * QTScheduledBandwidthPtr;

typedef QTScheduledBandwidthPtr * QTScheduledBandwidthHandle;

```

Special Considerations

The `OpaqueQTBandwidthReference` and `OpaqueQTScheduledBandwidthReference` data structures are not part of the public QuickTime API and are not documented.

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 11.

Music Types

Support the QuickTime Music Architecture.

```
typedef ComponentInstance MusicComponent;

typedef ComponentInstance TunePlayer;

typedef SInt32 MusicController;

typedef MusicDescription * MusicDescriptionPtr;

typedef MusicDescriptionPtr * MusicDescriptionHandle;

typedef ComponentInstance QTMIDIComponent;

typedef QTMIDIPortList * QTMIDIPortListPtr;

typedef QTMIDIPortListPtr * QTMIDIPortListHandle;

typedef GCInstrumentData * GCInstrumentDataPtr;

typedef GCInstrumentDataPtr * GCInstrumentDataHandle;

typedef GenericKnobDescriptionList * GenericKnobDescriptionListPtr;

typedef GenericKnobDescriptionListPtr * GenericKnobDescriptionListHandle;

typedef Handle AtomicInstrument;

typedef Ptr AtomicInstrumentPtr;

typedef InstrumentInfoList * InstrumentInfoListPtr;

typedef InstrumentInfoListPtr * InstrumentInfoListHandle;

typedef InstrumentAboutInfo * InstrumentAboutInfoPtr;

typedef InstrumentAboutInfoPtr * InstrumentAboutInfoHandle;

typedef GMInstrumentInfo * GMInstrumentInfoPtr;

typedef GMInstrumentInfoPtr * GMInstrumentInfoHandle;

typedef nonGMInstrumentInfo * nonGMInstrumentInfoPtr;

typedef nonGMInstrumentInfoPtr * nonGMInstrumentInfoHandle;

typedef InstCompInfo * InstCompInfoPtr;
```

```
typedef InstCompInfoPtr * InstCompInfoHandle;  
  
typedef ComponentInstance NoteAllocator;  
  
typedef struct OpaqueNoteChannel * NoteChannel;  
  
typedef unsigned long MusicOpWord;  
  
typedef MusicOpWord * MusicOpWordPtr;
```

Special Considerations

The `OpaqueNoteChannel` data structure is not part of the public QuickTime API and is not documented.

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 15.

Numeric Types

Define numeric values in QuickTime programming.

```
typedef unsigned char UInt8;  
  
typedef UInt8 Byte;  
  
typedef signed char SInt8;  
  
typedef SInt8 SignedByte;  
  
typedef SInt8 VHSelct;  
  
typedef unsigned short UInt16;  
  
typedef signed short SInt16;  
  
typedef unsigned long UInt32;  
  
typedef signed long SInt32;  
  
typedef short ShortFixed;  
  
typedef long Fixed;  
  
typedef Fixed * FixedPtr;  
  
typedef unsigned long UnsignedFixed;
```

```

typedef unsigned Fixed * unsignedFixedPtr;

typedef long Fract;

typedef Fract * FractPtr;

typedef wide SInt64;

typedef wide * WidePtr;

typedef unsignedWide UInt64;

typedef unsignedWide * unsignedWidePtr;

typedef float Float32;

typedef double Float64;

typedef long double Float96;

typedef long double Float80;

typedef Float32 QTFloatSingle;

typedef Float64 QTFloatDouble;

typedef Float80 extended80;

typedef Float96 extended96;

```

Sequence Grabbing Types

Support sequence grabbing components, including channel and panel components.

```

typedef ComponentInstance SeqGrabComponent;

typedef ComponentInstance SGChannel;

typedef unsigned long SeqGrabDataOutputEnum;

typedef unsigned long SeqGrabUsageEnum;

typedef unsigned long SeqGrabChannelInfoEnum;

typedef SGOutputRecord * SGOutput;

```

```
typedef SeqGrabFrameInfo * SeqGrabFrameInfoPtr;

typedef SeqGrabExtendedFrameInfo * SeqGrabExtendedFrameInfoPtr;

typedef SGDeviceListRecord * SGDeviceListPtr;

typedef SGDeviceListPtr * SGDeviceList;
```

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 12.

SMIL Types

Define XML types for QuickTime SMIL features.

```
typedef XMLDocRecord * XMLDoc;

typedef XMLAttribute * XMLAttributePtr;

union XMLAttributeValue {
    Sint32      number;
    Boolean     boolean;
    RGBColor    color;
    UInt32      enumType;
};

typedef XMLContent * XMLContentPtr;

typedef XMLElement * XMLElementPtr;

union XMLElementContent {
    XMLElement  element;
    char        *charData;
};
```

Sound Types

Provide support for the Sound Manager.

```
typedef SoundInfoList * SoundInfoListPtr;

typedef SoundComponentData * SoundComponentDataPtr;

typedef ExtendedSoundComponentData * ExtendedSoundComponentDataPtr;
```

```

typedef SoundDescription * SoundDescriptionPtr;
typedef SoundDescriptionPtr * SoundDescriptionHandle;
typedef SoundDescriptionV1 * SoundDescriptionV1Ptr;
typedef SoundDescriptionV1Ptr * SoundDescriptionV1Handle;
typedef SoundHeader * SoundHeaderPtr;
typedef CmpSoundHeader * CmpSoundHeaderPtr;
typedef ExtSoundHeader * ExtSoundHeaderPtr;
union SoundHeaderUnion {
    SoundHeader      stdHeader;
    CmpSoundHeader   cmpHeader;
    ExtSoundHeader   extHeader;
};
typedef ExtendedSoundParamBlock * ExtendedSoundParamBlockPtr;
typedef StateBlock * StateBlockPtr;
typedef LeftOverBlock * LeftOverBlockPtr;
typedef SndListResource * SndListPtr;
typedef SndListPtr * SndListHandle;
typedef SndListHandle SndListHndl;
typedef Snd2ListResource * Snd2ListPtr;
typedef Snd2ListPtr * Snd2ListHandle;
typedef Snd2ListHandle Snd2ListHndl;
typedef ConversionBlock * ConversionBlockPtr;
typedef ScheduledSoundHeader * ScheduledSoundHeaderPtr;
typedef ExtendedScheduledSoundHeader * ExtendedScheduledSoundHeaderPtr;
typedef SMStatus * SMStatusPtr;
typedef SCStatus * SCStatusPtr;
typedef AudioSelection * AudioSelectionPtr;

```



```

typedef CompressionInfo * CompressionInfoPtr;

typedef CompressionInfoPtr * CompressionInfoHandle;

typedef SoundSlopeAndInterceptRecord * SoundSlopeAndInterceptPtr;

typedef SoundSource * SoundSourcePtr;

typedef struct OpaqueSoundSource * SoundSource;

typedef struct OpaqueSoundConverter * SoundConverter;

typedef SoundComponentLink * SoundComponentLinkPtr;

typedef AudioInfo * AudioInfoPtr;

typedef AudioFormatAtom * AudioFormatAtomPtr;

typedef AudioEndianAtom * AudioEndianAtomPtr;

typedef AudioTerminatorAtom * AudioTerminatorAtomPtr;

typedef LevelMeterInfo * LevelMeterInfoPtr;

typedef EQSpectrumBandsRecord * EQSpectrumBandsRecordPtr;

typedef SPB * SPBPtr;

typedef SndDoubleBufferHeader * SndDoubleBufferHeaderPtr;

typedef SndDoubleBufferHeader2 * SndDoubleBufferHeader2Ptr;

typedef SndInputCmpParam * SndInputCmpParamPtr;

typedef SndChannel * SndChannelPtr;

typedef SndDoubleBuffer * SndDoubleBufferPtr;

typedef SoundParamBlock * SoundParamBlockPtr;

typedef const short * ActivationOrderListPtr;

typedef const OSType * ConstSFTTypeListPtr;

typedef struct SSpLocationData SSpLocationData;    // see SoundSprocket.h

typedef struct SSpVirtualSourceData SSpVirtualSourceData;
                                                // see SoundSprocket.h

```

Special Considerations

The `OpaqueSoundSource` and `OpaqueSoundConverter` data structures are not part of the public QuickTime API and are not documented.

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 13.

Sprite Management Types

Support QuickTime **sprite** programming.

```
typedef SpriteWorldRecord * SpriteWorld;

typedef SpriteRecord * Sprite;

typedef SpriteDescription * SpriteDescriptionPtr;

typedef SpriteDescriptionPtr * SpriteDescriptionHandle;

typedef QTRuntimeSpriteDescStruct * QTRuntimeSpriteDescPtr;

typedef QTSpriteButtonBehaviorStruct * QTSpriteButtonBehaviorPtr;
```

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 16.

Streaming Types

Support QuickTime streaming.

```
typedef QTSSampleDescription * QTSSampleDescriptionPtr;

typedef QTSSampleDescriptionPtr * QTSSampleDescriptionHandle;

typedef UInt32 RTPMPSampleRef;

typedef UInt32 RTPSSRC;

typedef RTPPayloadSortRequest * RTPPayloadSortRequestPtr;

typedef RTPPayloadInfo * RTPPayloadInfoPtr;

typedef RTPPayloadInfoPtr * RTPPayloadInfoHandle;

typedef ComponentInstance RTPReassembler;
```

```
typedef RTPReassemblerInfo * RTPReassemblerInfoPtr;

typedef RTPReassemblerInfoPtr * RTPReassemblerInfoHandle;

typedef ComponentInstance RTPMediaPacketizer;

typedef struct OpaqueRTPPacketGroupRef * RTPPacketGroupRef;

typedef struct OpaqueRTPPacketRef * RTPPacketRef;

typedef struct OpaqueRTPPacketRepeatedDataRef * RTPPacketRepeatedDataRef;

typedef MediaPacketizerRequirements * MediaPacketizerRequirementsPtr;

typedef MediaPacketizerInfo * MediaPacketizerInfoPtr;

typedef MediaPacketizerInfoPtr * MediaPacketizerInfoHandle;

typedef ComponentInstance RTPPacketBuilder;

typedef QTSPresentationRecord * QTSPresentation;

typedef QTSSStreamRecord * QTSSStream;

typedef QTSEditList * QTSEditListPtr;

typedef QTSEditListPtr * QTSEditListHandle;

typedef struct OpaqueQTSMemPtr * QTSMemPtr;

typedef QTSSStatHelperRecord * QTSSStatHelper;

typedef UInt32 QTSSStatus;
```

Special Considerations

The `OpaqueRTPPacketGroupRef`, `OpaqueRTPPacketRef`, `OpaqueQTSMemPtr`, and `OpaqueRTPPacketRepeatedDataRef` data structures are not part of the public QuickTime API and are not documented.

String Types

Represent string values in QuickTime.

```
typedef char * CharsPtr;

typedef CharsPtr * CharsHandle;
```

```
typedef unsigned char * StringPtr;

typedef StringPtr * StringHandle;

typedef char Chars;

typedef unsigned char Str255;

typedef unsigned char Str63;

typedef unsigned char Str32;

typedef unsigned char Str31;

typedef unsigned char Str27;

typedef unsigned char Str15;

typedef unsigned char Str32Field;

typedef const unsigned char * ConstStringPtr;

typedef const unsigned char * ConstStr255Param;

typedef const unsigned char * ConstStr63Param;

typedef const unsigned char * ConstStr32Param;

typedef const unsigned char * ConstStr31Param;

typedef const unsigned char * ConstStr27Param;

typedef const unsigned char * ConstStr15Param;

typedef UInt16 UniChar;

typedef UInt32 UniCharCount;

typedef Str255 StrFileName;

typedef ConstStr255Param ConstStrFileNameParam;
```

System and Toolbox Types

Define basic data formats for the QuickTime system software.

```

typedef SInt16 OSErr;

typedef SInt32 OSStatus;

typedef unsigned long FourCharCode;

typedef FourCharCode OSType;

typedef FourCharCode ResType;

typedef OSType * OSTypePtr;

typedef ResType * ResTypePtr;

typedef UInt32 OptionBits;

typedef UInt32 ItemCount;

typedef UInt32 PBVersion;

typedef MenuInfo * MenuPtr;

typedef MenuPtr * MenuHandle;

typedef UInt16 EventKind;

typedef UInt16 EventMask;

typedef UInt16 EventModifiers;

typedef DialogPtr DialogRef;

typedef unsigned long ProcInfoType;

typedef SProcRec * SProcPtr;

typedef SProcPtr * SProcHndl;

typedef UInt8 RDFlagsType;                // in MixedMode.h

typedef SInt8 ISAType;                    // in MixedMode.h

typedef unsigned short RoutineFlagsType;  // in MixedMode.h

typedef ComponentMPWorkFunctionHeaderRecord *
                                           ComponentMPWorkFunctionHeaderRecordPtr;

typedef SInt16 DialogItemIndex;           // in Dialogs.h

```

Text Types

Support QuickTime text components.

```
typedef TextDescription * TextDescriptionPtr;

typedef TextDescriptionPtr * TextDescriptionHandle;

typedef TERec * TEPtr;

typedef TEPtr * TEHandle;

typedef TCTextOptions * TCTextOptionsPtr;

typedef SInt16 ScriptCode;

typedef SInt16 LangCode;

typedef SInt16 RegionCode;

typedef struct STElement STElement;    // see TextEdit.h
typedef STElement * STPtr;              // in TextEdit.h
typedef STPtr * STHandle;                // in TextEdit.h
typedef struct LHElement LHElement;    // see TextEdit.h
typedef LHElement * LHPtr;              // in TextEdit.h
typedef LHPtr * LHHandle;                // in TextEdit.h
typedef struct NullStRec NullStRec;       // see TextEdit.h
typedef NullStRec * NullStPtr;           // in TextEdit.h
typedef NullStPtr * NullStHandle;        // in TextEdit.h
typedef struct StyleRun StyleRun;        // see TextEdit.h
```

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 6.

Timing and Callback Types

Support QuickTime's timing and callback capabilities.

```

typedef long TimeValue;

typedef long TimeScale;

typedef SInt64 TimeValue64;

typedef UnsignedWide AbsoluteTime;

typedef wide CompTimeValue;

typedef SInt32 Duration;

typedef TimeBaseRecord * TimeBase;

typedef unsigned long TimeBaseFlags;

typedef unsigned short nextTimeFlagsEnum;

typedef unsigned long TimeBaseStatus;

typedef TimeCodeDescription * TimeCodeDescriptionPtr;

typedef TimeCodeDescriptionPtr * TimeCodeDescriptionHandle;

typedef unsigned short QTCallbackType;

typedef CallbackRecord * QTCallback;

typedef unsigned short QTCallbackFlags;

typedef QTSyncTaskRecord * QTSyncTaskPtr;

```

Universal Procedure Pointers

Support the Macintosh Mixed Mode Manager.

```

typedef ProcPtr UniversalProcPtr;

typedef ProcPtr * ProcHandle;

typedef struct RoutineDescriptor * UniversalProcPtr;

typedef UniversalProcPtr * UniversalProcHandle;

typedef STACK_UPP_TYPE(DoMCActionProcPtr) DoMCActionUPP;

typedef STACK_UPP_TYPE(MovieExecuteWiredActionsProcPtr)

```

```

MovieExecuteWiredActionsUPP;

typedef STACK_UPP_TYPE(MoviePrePrerollCompleteProcPtr)
MoviePrePrerollCompleteUPP;

typedef STACK_UPP_TYPE(SequenceFilterDataProcPtr) SequenceFilterDataUPP;

typedef STACK_UPP_TYPE(SFModalFilterProcPtr) SFModalFilterUPP;

typedef STACK_UPP_TYPE(SGDataProcPtr) SGDataUPP;

typedef STACK_UPP_TYPE(QTVRInterceptProcPtr) QTVRInterceptUPP;

typedef STACK_UPP_TYPE(SoundConverterFillBufferDataProcPtr)
SoundConverterFillBufferDataUPP;

typedef STACK_UPP_TYPE(DlgHookProcPtr) DlgHookUPP;

typedef STACK_UPP_TYPE(FileFilterProcPtr) FileFilterUPP;

typedef STACK_UPP_TYPE(DlgHookYDProcPtr) DlgHookYDUPP;

typedef STACK_UPP_TYPE(ModalFilterYDProcPtr) ModalFilterYDUPP;

typedef STACK_UPP_TYPE(FileFilterYDProcPtr) FileFilterYDUPP;

typedef STACK_UPP_TYPE(ActivateYDProcPtr) ActivateYDUPP;

typedef STACK_UPP_TYPE(ModalFilterProcPtr) ModalFilterUPP;

typedef STACK_UPP_TYPE(ImageCodecMPDrawBandProcPtr) ImageCodecMPDrawBandUPP;

typedef STACK_UPP_TYPE(ICMDataProcPtr) ICMDataUPP;

typedef STACK_UPP_TYPE(ICMFlushProcPtr) ICMFlushUPP;

typedef STACK_UPP_TYPE(ICMCompletionProcPtr) ICMCompletionUPP;

typedef STACK_UPP_TYPE(ICMProgressProcPtr) ICMProgressUPP;

typedef STACK_UPP_TYPE(StdPixProcPtr) StdPixUPP;

typedef STACK_UPP_TYPE(QDPixProcPtr) QDPixUPP;

typedef STACK_UPP_TYPE(ICMAalignmentProcPtr) ICMAalignmentUPP;

typedef STACK_UPP_TYPE(ICMCursorShieldedProcPtr) ICMCursorShieldedUPP;

typedef STACK_UPP_TYPE(ICMMemoryDisposedProcPtr) ICMMemoryDisposedUPP;

```



```
typedef STACK_UPP_TYPE(ICMConvertDataFormatProcPtr) ICMConvertDataFormatUPP;  
typedef STACK_UPP_TYPE(PrePrerollCompleteProcPtr) PrePrerollCompleteUPP;  
typedef STACK_UPP_TYPE(MovieRgnCoverProcPtr) MovieRgnCoverUPP;  
typedef STACK_UPP_TYPE(MovieProgressProcPtr) MovieProgressUPP;  
typedef STACK_UPP_TYPE(MovieDrawingCompleteProcPtr) MovieDrawingCompleteUPP;  
typedef STACK_UPP_TYPE(TrackTransferProcPtr) TrackTransferUPP;  
typedef STACK_UPP_TYPE(GetMovieProcPtr) GetMovieUPP;  
typedef STACK_UPP_TYPE(MoviePreviewCallOutProcPtr) MoviePreviewCallOutUPP;  
typedef STACK_UPP_TYPE(TextMediaProcPtr) TextMediaUPP;  
typedef STACK_UPP_TYPE(ActionsProcPtr) ActionsUPP;  
typedef STACK_UPP_TYPE(MCActionFilterProcPtr) MCActionFilterUPP;  
typedef STACK_UPP_TYPE(MCActionFilterWithRefConProcPtr)  
MCActionFilterWithRefConUPP;  
typedef STACK_UPP_TYPE(QTCallbackProcPtr) QTCallbackUPP;  
typedef STACK_UPP_TYPE(QTSyncTaskProcPtr) QTSyncTaskUPP;  
typedef STACK_UPP_TYPE(TweenerDataProcPtr) TweenerDataUPP;  
typedef STACK_UPP_TYPE(DataHCompletionProcPtr) DataHCompletionUPP;  
typedef STACK_UPP_TYPE(MovieExportGetDataProcPtr) MovieExportGetDataUPP;  
typedef STACK_UPP_TYPE(MovieExportGetPropertyProcPtr)  
MovieExportGetPropertyUPP;  
typedef STACK_UPP_TYPE(SCModalFilterProcPtr) SCModalFilterUPP;  
typedef STACK_UPP_TYPE(SCModalHookProcPtr) SCModalHookUPP;  
typedef STACK_UPP_TYPE(SGGrabBottleProcPtr) SGGrabBottleUPP;  
typedef STACK_UPP_TYPE(SGGrabCompleteBottleProcPtr) SGGrabCompleteBottleUPP;  
typedef STACK_UPP_TYPE(SGDisplayBottleProcPtr) SGDisplayBottleUPP;
```

```

typedef STACK_UPP_TYPE(SGCompressBottleProcPtr) SGCompressBottleUPP;

typedef STACK_UPP_TYPE(SGCompressCompleteBottleProcPtr)
SGCompressCompleteBottleUPP;

typedef STACK_UPP_TYPE(SGAddFrameBottleProcPtr) SGAddFrameBottleUPP;

typedef STACK_UPP_TYPE(SGTransferFrameBottleProcPtr)
SGTransferFrameBottleUPP;

typedef STACK_UPP_TYPE(SGGrabCompressCompleteBottleProcPtr)
SGGrabCompressCompleteBottleUPP;

typedef STACK_UPP_TYPE(SGDisplayCompressBottleProcPtr)
SGDisplayCompressBottleUPP;

typedef STACK_UPP_TYPE(SGModalFilterProcPtr) SGModalFilterUPP;

typedef STACK_UPP_TYPE(VdigIntProcPtr) VdigIntUPP;

typedef STACK_UPP_TYPE(MusicMIDISendProcPtr) MusicMIDISendUPP;

typedef STACK_UPP_TYPE(MusicMIDIReadHookProcPtr) MusicMIDIReadHookUPP;

typedef STACK_UPP_TYPE(MusicOfflineDataProcPtr) MusicOfflineDataUPP;

typedef STACK_UPP_TYPE(TuneCallBackProcPtr) TuneCallBackUPP;

typedef STACK_UPP_TYPE(TunePlayCallBackProcPtr) TunePlayCallBackUPP;

typedef STACK_UPP_TYPE(QTVRLeavingNodeProcPtr) QTVRLeavingNodeUPP;

typedef STACK_UPP_TYPE(QTVREnteringNodeProcPtr) QTVREnteringNodeUPP;

typedef STACK_UPP_TYPE(QTVRMouseOverHotSpotProcPtr) QTVRMouseOverHotSpotUPP;

typedef STACK_UPP_TYPE(QTVRImagingCompleteProcPtr) QTVRImagingCompleteUPP;

typedef STACK_UPP_TYPE(QTVRBackBufferImagingProcPtr)
QTVRBackBufferImagingUPP;

typedef STACK_UPP_TYPE(FilePlayCompletionProcPtr) FilePlayCompletionUPP;

typedef STACK_UPP_TYPE(SICompletionProcPtr) SICompletionUPP;

typedef STACK_UPP_TYPE(SndCallBackProcPtr) SndCallBackUPP;

typedef STACK_UPP_TYPE(SndDoubleBackProcPtr) SndDoubleBackUPP;

```

```

typedef STACK_UPP_TYPE(SoundParamProcPtr) SoundParamUPP;

typedef STACK_UPP_TYPE(ComponentMPWorkFunctionProcPtr)
ComponentMPWorkFunctionUPP;

typedef STACK_UPP_TYPE(ComponentRoutineProcPtr) ComponentRoutineUPP;

typedef STACK_UPP_TYPE(GetMissingComponentResourceProcPtr)
GetMissingComponentResourceUPP;

typedef STACK_UPP_TYPE(RTPMPDataReleaseProcPtr) RTPMPDataReleaseUPP;

typedef STACK_UPP_TYPE(RTPPBCallbackProcPtr) RTPPBCallbackUPP;

typedef STACK_UPP_TYPE(MoviesErrorProcPtr) MoviesErrorUPP;

typedef STACK_UPP_TYPE(QTSNotificationProcPtr) QTSNotificationUPP;

typedef STACK_UPP_TYPE(ImageCodecTimeTriggerProcPtr)
ImageCodecTimeTriggerUPP;

typedef STACK_UPP_TYPE(QTBandwidthNotificationProcPtr)
QTBandwidthNotificationUPP;

typedef STACK_UPP_TYPE(QDTextProcPtr) QDTextUPP;

typedef STACK_UPP_TYPE(QDLineProcPtr) QDLineUPP;

typedef STACK_UPP_TYPE(QDRectProcPtr) QDRectUPP;

typedef STACK_UPP_TYPE(QDRRectProcPtr) QDRRectUPP;

typedef STACK_UPP_TYPE(QDOvalProcPtr) QDOvalUPP;

typedef STACK_UPP_TYPE(QDArcProcPtr) QDArcUPP;

typedef STACK_UPP_TYPE(QDPolyProcPtr) QDPolyUPP;

typedef STACK_UPP_TYPE(QDRgnProcPtr) QDRgnUPP;

typedef STACK_UPP_TYPE(QDBitsProcPtr) QDBitsUPP;

typedef STACK_UPP_TYPE(QDCommentProcPtr) QDCommentUPP;

typedef STACK_UPP_TYPE(QDTxMeasProcPtr) QDTxMeasUPP;

typedef STACK_UPP_TYPE(QDGetPicProcPtr) QDGetPicUPP;

typedef STACK_UPP_TYPE(QDPutPicProcPtr) QDPutPicUPP;

```

```

typedef STACK_UPP_TYPE(QDOPcodeProcPtr) QDOPcodeUPP;

typedef STACK_UPP_TYPE(QDStdGlyphsProcPtr) QDStdGlyphsUPP;

typedef STACK_UPP_TYPE(QDJShieldCursorProcPtr) QDJShieldCursorUPP;

typedef STACK_UPP_TYPE(DragGrayRgnProcPtr) DragGrayRgnUPP;

typedef STACK_UPP_TYPE(ColorSearchProcPtr) ColorSearchUPP;

typedef STACK_UPP_TYPE(ColorComplementProcPtr) ColorComplementUPP;

typedef STACK_UPP_TYPE(QDPrinterStatusProcPtr) QDPrinterStatusUPP;

typedef REGISTER_UPP_TYPE(HighHookProcPtr) HighHookUPP;

typedef REGISTER_UPP_TYPE(CaretHookProcPtr) CaretHookUPP;

typedef REGISTER_UPP_TYPE(TEClickLoopProcPtr) TEClickLoopUPP;

typedef REGISTER_UPP_TYPE(WordBreakProcPtr) WordBreakUPP;

typedef REGISTER_UPP_TYPE(SIInterruptProcPtr) SIInterruptUPP;

```

See Also

See *Inside Macintosh: PowerPC System Software* (listed in the **bibliography**).

Video Types

Support QuickTime video components.

```

typedef ComponentInstance VideoDigitizerComponent;

typedef ComponentInstance QTVideoOutputComponent;

typedef ComponentResult VideoDigitizerError;

typedef VDCompressionList * VDCompressionListPtr;

typedef VDCompressionListPtr * VDCompressionListHandle;

typedef VdigBufferRecList * VdigBufferRecListPtr;

typedef VdigBufferRecListPtr * VdigBufferRecListHandle;

```

Virtual Reality Types

Support the programming interface for QuickTime Virtual Reality.

```
typedef struct OpaqueQTVRInstance * QTVRInstance;

typedef UInt32 QTVRProcSelector;

typedef QTVRInterceptRecord * QTVRInterceptPtr;

typedef UInt32 QTVRImagingMode;

typedef UInt32 QTVRAngularUnits;

typedef UInt32 QTVRObjectAnimationSetting;

typedef UInt32 QTVRControlSetting;

typedef UInt32 QTVRViewStateType;

typedef UInt32 QTVRNudgeControl;

typedef UInt32 QTVRNudgeMode;

typedef QTVRCursorRecord CursorRecord;

typedef QTVRAreaOfInterest AreaOfInterest;

typedef QTVRFloatPoint FloatPoint;

typedef QTVRLeavingNodeProcPtr LeavingNodeProcPtr;

typedef QTVRLeavingNodeUPP LeavingNodeUPP;

typedef QTVREnteringNodeProcPtr EnteringNodeProcPtr;

typedef QTVREnteringNodeUPP EnteringNodeUPP;

typedef QTVRMouseOverHotSpotProcPtr MouseOverHotSpotProcPtr;

typedef QTVRMouseOverHotSpotUPP MouseOverHotSpotUPP;

typedef QTVRImagingCompleteProcPtr ImagingCompleteProcPtr;

typedef QTVRImagingCompleteUPP ImagingCompleteUPP;

typedef QTVRBackBufferImagingProcPtr BackBufferImagingProcPtr;

typedef QTVRBackBufferImagingUPP BackBufferImagingUPP;
```

TYP

Defined Data Types

Special Considerations

The `OpaqueQTVRInstance` data structure is not part of the public QuickTime API and is not documented.

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 17.

Windows Programming Types

Support QuickTime on the Windows platform.

```
typedef long QTMLMutex;  
  
typedef struct OpaqueQTMLSyncVar * QTMLSyncVar;  
  
typedef QTMLSyncVar * QTMLSyncVarPtr;
```

Special Considerations

The `OpaqueQTMLSyncVar` data structure is not part of the public QuickTime API and is not documented.

See Also

See *Discovering QuickTime* (listed in the **bibliography**), Chapter 8.

Constants

Atom ID Codes

Identify the four-character type codes of atoms.

BandwidthManagementPrefsType	= 'bwmg'
ClipAID	= 'clip'
ColorTableAID	= 'ctab'
CompressedMovieAID	= 'cmov'
CompressedMovieDataAID	= 'cmvd'
ConnectionSpeedPrefsType	= 'cspd'
DataCompressionAtomAID	= 'dcom'
DataInfoAID	= 'dinf'
DataRefAID	= 'dref'
DataRefContainerAID	= 'drfc'
EditListAID	= 'elst'
EditsAID	= 'edts'
FreeAtomType	= 'free'
GenericMediaInfoAID	= 'gmin'
GenericMediaInfoHeaderAID	= 'gmhd'
HandlerAID	= 'hdlr'
InputMapAID	= 'imap'
kAction	= 'actn'
kActionFlags	= 'flag'
kActionListAtomType	= 'list'
kActionParameter	= 'parm'
kActionParameterMaxValue	= 'maxv'
kActionParameterMinValue	= 'minv'
kActionTarget	= 'targ'
kAudioEndianAtomType	= 'enda'
kAudioFormatAtomType	= 'frma'
kAudioTerminatorAtomType	= 0x00000000
kAudioVBRAtomType	= 'vbra'
kCommentAtomType	= 'why '
kConditionalAtomType	= 'test'
kCustomActionHandler	= 'cust'
kCustomHandlerDesc	= 'desc'
kCustomHandlerID	= 'id '
kEffectManufacturerAtom	= 'manu'
kEffectNameAtom	= 'name'
kEffectTypeAtom	= 'type'
kExpressionContainerAtomType	= 'expr'
kGraphicsExportDescription	= 'desc'

kGraphicsExportExtension	= 'ext '
kGraphicsExportFileType	= 'ftyp'
kGraphicsExportGroup	= 'expo'
kGraphicsExportMIMETYPE	= 'mime'
kInitialRotationAtom	= 'intro'
kInputMapSubInputID	= 'subi'
kITextAtomType	= 'itxt'
kITextStringAtomType	= 'text'
kListElementDataType	= 'daty'
kListElementType	= 'type'
kMovieMediaAltText	= 'altt'
kMovieMediaAutoPlay	= 'play'
kMovieMediaClipBegin	= 'clpb'
kMovieMediaClipDuration	= 'clpd'
kMovieMediaDataReference	= 'mmdr'
kMovieMediaEnableFrameStepping	= 'enfs'
kMovieMediaHeight	= 'ht '
kMovieMediaLeft	= 'left'
kMovieMediaLoop	= 'loop'
kMovieMediaRectangleAtom	= 'rect'
kMovieMediaRegionAtom	= 'regi'
kMovieMediaSlaveAudio	= 'slau'
kMovieMediaSlaveTrackDuration	= 'sltr'
kMovieMediaSpatialAdjustment	= 'fit '
kMovieMediaTitle	= 'titl'
kMovieMediaTop	= 'top '
kMovieMediaUseMIMETYPE	= 'mime'
kMovieMediaWidth	= 'wd '
kNameAtom	= 'name'
kNonLinearTweenHeader	= 'nlth'
kOperandAtomType	= 'oprn'
kOperatorAtomType	= 'oper'
kQTCOLORSyncProfile	= 'iccp'
kQTDontRecompress	= 'dntr'
kQTEventRecordAtomType	= 'erec'
kQTEventType	= 'evnt'
kQTInterlaceStyle	= 'ilac'
kQTParseTextHREFBaseURL	= 'burl'
kQTParseTextHREFClickPoint	= 'clik'
kQTParseTextHREFDelimiter	= 'delm'
kQTParseTextHREFHREF	= 'href'
kQTParseTextHREFIsAutoHREF	= 'auto'
kQTParseTextHREFRecomposeHREF	= 'rhrf'
kQTParseTextHREFTarget	= 'targ'
kQTParseTextHREFUseAltDelim	= 'altd'
kQTResolutionSettings	= 'reso'
kQTSAnnotationsChangedNotification	= 'meta'
kQTSBandwidthAlertNotification	= 'bwal'
kQTSBufferTimeAID	= 'bufr'

kQTSClipRectAID	= 'clip'
kQTSConnectionAtomType	= 'conn'
kQTSConnectionMethodPrefsType	= 'mthd'
kQTSConnectionPrefsType	= 'stcm'
kQTSConnectionPrefsVersion	= 'vers'
kQTSDirectConnectPrefsType	= 'drct'
kQTSDontProxyDataType	= 'data'
kQTSDontProxyPrefsAtomType	= 'nopr'
kQTSDurationAID	= 'dura'
kQTSDurationInfo	= 'dura'
kQTSErrorNotification	= 'err '
kQTSHTTProxyPrefsType	= 'http'
kQTSHTTTransportType	= 'http'
kQTSLostPercentInfo	= 'lpct'
kQTSMediaDescriptionTextAID	= 'mdes'
kQTSMediaStreamAID	= 'mstr'
kQTSMediaStreamHeaderAID	= 'mshd'
kQTSMediaTypeInfo	= 'mtyp'
kQTSTNameInfo	= 'name'
kQTSTNewPresentationNotification	= 'nprs'
kQTSTNormalStatusDimensionsInfo	= 'nstd'
kQTSTPrerollAckNotification	= 'pack'
kQTSTPresDoneChangingNotification	= 'prcd'
kQTSTPresentationAID	= 'pres'
kQTSTPresentationChangedNotification	= 'prch'
kQTSTPresentationGoneNotification	= 'xprs'
kQTSTPresentationHeaderAID	= 'phdr'
kQTSTProxyPrefsAtomType	= 'prxy'
kQTSTRTSPProxyPrefsType	= 'rtsp'
kQTSTServerNameNotification	= 'snam'
kQTSTSOCKSPrefsType	= 'sock'
kQTSTSOCKSProxyPrefsType	= 'scks'
kQTSTSourceClipRectInfo	= 'oclp'
kQTSTSourceGraphicsModeInfo	= 'ogrm'
kQTSTSourceLanguageInfo	= 'olng'
kQTSTSourceMatrixInfo	= 'omat'
kQTSTSourceTrackFlagsInfo	= 'otfl'
kQTSTSourceUserDataInfo	= 'oudt'
kQTSTStartAckNotification	= 'sack'
kQTSTStatisticsInfo	= 'stat'
kQTSTStatisticsNameAtomType	= 'name'
kQTSTStatisticsStreamAtomType	= 'strm'
kQTSTStatisticsUnitsAtomType	= 'unit'
kQTSTStatisticsUnitsNameAtomType	= 'unin'
kQTSTStatusNotification	= 'stat'
kQTSTStreamBeginChangingNotification	= 'stcb'
kQTSTStreamDoneChangingNotification	= 'stcd'
kQTSTStreamMediaType	= 'strm'
kQTSTTargetBufferDurationInfo	= 'bufr'

kQTSTCPTransportType	= 'tcp '
kQTSTotalDataRateInfo	= 'drtt'
kQTSTotalDataRateInInfo	= 'drti'
kQTSTotalDataRateOutInfo	= 'drto'
kQTSTransAndProxyAtomType	= 'strp'
kQTSTransportPrefsAtomType	= 'trns'
kQTSUDPTransportType	= 'udp '
kQTTTargetDataSize	= 'dasz'
kQTVRAngleRangeAtomType	= 'arng'
kQTVRColorCursorAtomType	= 'crsr'
kQTVRCursorAtomType	= 'CURS'
kQTVRCursorParentAtomType	= 'vrcp'
kQTVRFOVConstraintAtomType	= 'fcon'
kQTVRHotSpotAtomType	= 'hots'
kQTVRHotSpotInfoAtomType	= 'hsin'
kQTVRHotSpotParentAtomType	= 'hspa'
kQTVRHotSpotTrackRefAtomType	= 'hstr'
kQTVRImageTrackRefAtomType	= 'imtr'
kQTVRImagingParentAtomType	= 'imgp'
kQTVRLinkInfoAtomType	= 'link'
kQTVRNodeHeaderAtomType	= 'ndhd'
kQTVRNodeIDAtomType	= 'vrni'
kQTVRNodeLocationAtomType	= 'nloc'
kQTVRNodeParentAtomType	= 'vrnp'
kQTVRObjectHotSpotTrackRefAtomID	= 0x00000001
kQTVRObjectImageTrackRefAtomID	= 0x00000001
kQTVRObjectImagingAtomType	= 'imob'
kQTVRObjectInfoAtomID	= 0x00000001
kQTVRObjectInfoAtomType	= 'obji'
kQTVRPanConstraintAtomType	= 'pcon'
kQTVRPanoImagingAtomType	= 'impn'
kQTVRPanoSampleDataAtomType	= 'pdat'
kQTVRStringAtomType	= 'vrsg'
kQTVRStringEncodingAtomType	= 'vrse'
kQTVRTiltConstraintAtomType	= 'tcon'
kQTVRTrackRefArrayAtomType	= 'tref'
kQTVRWorldHeaderAtomType	= 'vrsc'
kSpriteAtomType	= 'sprt'
kSpriteBehaviorsAtomType	= 'beha'
kSpriteCursorBehaviorAtomType	= 'crsr'
kSpriteFloatingPointVariableAtomType	= 'flov'
kSpriteImageAtomType	= 'imag'
kSpriteImageBehaviorAtomType	= 'imag'
kSpriteImageDataAtomType	= 'imda'
kSpriteImageDataRefAtomType	= 'imre'
kSpriteImageDataRefTypeAtomType	= 'imrt'
kSpriteImageDefaultImageIndexAtomType	= 'defi'
kSpriteImageGroupIDAtomType	= 'imgr'
kSpriteImageNameAtomType	= 'name'

```

kSpriteImageRegistrationAtomType      = 'imrg'
kSpriteImagesContainerAtomType        = 'imct'
kSpriteNameAtomType                   = 'name'
kSpriteSharedDataAtomType             = 'dflt'
kSpriteStatusStringsBehaviorAtomType  = 'sstr'
kSpriteStringVariableAtomType         = 'strv'
kSpriteUsesImageIDsAtomType           = 'uses'
kSpriteVariablesContainerAtomType      = 'vars'
kTargetChildMovieMovieID              = 'momi'
kTargetChildMovieTrackID              = 'moti'
kTargetChildMovieTrackName            = 'motn'
kTargetMovieID                        = 'moid'
kTargetParentMovie                    = 'mopa'
kTargetRootMovie                      = 'moro'
kTargetSpriteIndex                    = 'spin'
kTargetTrackID                        = 'trid'
kTargetTrackName                      = 'trna'
kTrackModifierInput                   = ' in'
kTrackModifierInputName               = 'name'
kTrackModifierObjectID                = 'obid'
kTrackModifierObjectQTEventSend       = 'evnt'
kTrackModifierPanAngle                = 'pan '
kTrackModifierReference                = 'ssrc'
kTrackModifierTiltAngle               = 'tilt'
kTrackModifierType                    = ' ty'
kTrackModifierVerticalFieldOfViewAngle = 'fov '
kTrackPropertyInstantiation            = 'inst'
kTrackPropertyMediaType                = 'mtyp'
kTrackReferenceChapterList             = 'chap'
kTrackReferenceModifier                = 'ssrc'
kTrackReferenceTimeCode                = 'tmcd'
kTween3dInitialCondition               = 'icnd'
kTweenData                             = 'data'
kTweenDuration                         = 'twdu'
kTweenEntry                            = 'twen'
kTweenFlags                            = 'flag'
kTweenInterpolationID                 = 'intr'
kTweenOutputMax                       = 'omax'
kTweenOutputMin                       = 'omin'
kTweenPictureData                     = 'PICT'
kTweenRegionData                      = 'qdrg'
kTweenSequenceElement                  = 'seqe'
kTweenStartOffset                     = 'twst'
kTweenType                             = 'twnt'
kTweenType3dAngleAspectCameraData     = '3caa'
kTweenType3dCameraData                 = '3cam'
kTweenType3dMatrix                     = '3mat'
kTweenType3dMatrixNonLinear            = '3nlr'
kTweenType3dQuaternion                 = '3qua'

```

kTweenType3dRotate	= '3rot'
kTweenType3dRotateAboutAxis	= '3rax'
kTweenType3dRotateAboutPoint	= '3rap'
kTweenType3dRotateAboutVector	= '3rvc'
kTweenType3dScale	= '3sca'
kTweenType3dSoundLocalizationData	= '3slc'
kTweenType3dTranslate	= '3tra'
kTweenType3dVRObject	= '3vro'
kTweenTypeAtomList	= 'atom'
kTweenTypeMultiMatrix	= 'mulm'
kTweenTypePathToFixedPoint	= 'gxfp'
kTweenTypePathToMatrixRotation	= 'gxpr'
kTweenTypePathToMatrixTranslation	= 'gxmt'
kTweenTypePathToMatrixTranslationAndRotation	= 'gxmr'
kTweenTypePathXtoY	= 'gxyy'
kTweenTypePathYtoX	= 'gxyx'
kTweenTypePolygon	= 'poly'
kTweenTypeSpin	= 'spin'
kWhichAction	= 'whic'
LoadSettingsAID	= 'load'
MatteAID	= 'matt'
MatteCompAID	= 'kmat'
MediaAID	= 'mdia'
MediaHeaderAID	= 'mdhd'
MediaInfoAID	= 'minf'
MovieAID	= 'moov'
MovieBufferHintsAID	= 'mbfh'
MovieDataAtomType	= 'mdat'
MovieDataRefAliasAID	= 'mdra'
MovieHeaderAID	= 'mvhd'
MovieResourceAtomType	= 'moov'
PropertyAtomAID	= 'code'
quickTimeImageFileColorSyncProfileAtom	= 'iicc'
quickTimeImageFileImageDataAtom	= 'idat'
quickTimeImageFileImageDescriptionAtom	= 'idsc'
quickTimeImageFileMetaDataAtom	= 'meta'
ReferenceMovieAlternateGroupAID	= 'rmag'
ReferenceMovieComponentCheckAID	= 'rmcd'
ReferenceMovieCPURatingAID	= 'rmcs'
ReferenceMovieDataRateAID	= 'rmdr'
ReferenceMovieDataRefAID	= 'rdrf'
ReferenceMovieDescriptorAID	= 'rmda'
ReferenceMovieLanguageAID	= 'rmla'
ReferenceMovieQualityAID	= 'rmqu'
ReferenceMovieRecordAID	= 'rmra'
ReferenceMovieVersionCheckAID	= 'rmvc'
RgnClipAID	= 'crgn'
SampleTableAID	= 'stbl'
scSpatialSettingsType	= 'sptl'

SkipAtomType	= 'skip'
SoundLocalizationAID	= 'sloc'
SoundMediaInfoHeaderAID	= 'smhd'
STChunkOffset64AID	= 'co64'
STChunkOffsetAID	= 'stco'
STSampleDescAID	= 'stsd'
STSampleIDAID	= 'stid'
STSampleSizeAID	= 'stsz'
STSampleToChunkAID	= 'stsc'
STShadowSyncAID	= 'stsh'
STSyncSampleAID	= 'stss'
STTimeToSampAID	= 'stts'
TrackAID	= 'trak'
TrackHeaderAID	= 'tkhd'
TrackReferenceAID	= 'tref'
UserDataAID	= 'udta'
VideoMediaInfoHeaderAID	= 'vmhd'
WideAtomPlaceholderType	= 'wide'

See Also

The atoms used in the QuickTime API are listed in the chapter “Atoms” in this book. For further information about atoms, see *Inside QuickTime: QuickTime File Format* (listed in the **bibliography**).

Codec Identifiers

Identify codec components and data types in QuickTime.

kAnimationCodecType	= 'rle '
kAVRJPEGCodecType	= 'avr '
kBaseCodecType	= 'base'
kBMPCCodecType	= 'WRLE'
kCinepakCodecType	= 'cvid'
kCloudCodecType	= 'clou'
kCMYKCodecType	= 'cmyk'
kComponentVideoCodecType	= 'yuv2'
kComponentVideoSigned	= 'yuvu'
kComponentVideoUnsigned	= 'yuvs'
kDVCMNTSCCodecType	= 'dvc '
kDVCPALCodecType	= 'dvcp'
kDVCPProNTSCCodecType	= 'dvpn'
kDVCPProPALCodecType	= 'dvpp'
kFireCodecType	= 'fire'
kFLCCodecType	= 'flic'
k48RGBCodecType	= 'b48r'
kGIFCodecType	= 'gif '
kGraphicsCodecType	= 'smc '

kH261CodecType	= 'h261'
kH263CodecType	= 'h263'
kIndeo4CodecType	= 'IV41'
kJPEGCodecType	= 'jpeg'
kMacPaintCodecType	= 'PNTG'
kMicrosoftVideo1CodecType	= 'msvc'
kMotionJPEGACodecType	= 'mjpa'
kMotionJPEGBCodecType	= 'mjpb'
kMpegYUV420CodecType	= 'myuv'
kOpenDMLJPEGCodecType	= 'dmb1'
kPhotoCDCodecType	= 'kpcd'
kPlanarRGBCodecType	= '8BPS'
kPNGCodecType	= 'png '
kQuickDrawCodecType	= 'qdrw'
kQuickDrawGXCodecType	= 'qdgx'
kRawCodecType	= 'raw '
kSGICodecType	= '.SGI'
k16GrayCodecType	= 'b16g'
k64ARGBCodecType	= 'b64a'
kSorensonCodecType	= 'SVQ1'
kSorensonYUV9CodecType	= 'syv9'
kTargaCodecType	= 'tga '
k32AlphaGrayCodecType	= 'b32a'
kTIFFCodecType	= 'tiff'
kVectorCodecType	= 'path'
kVideoCodecType	= 'rpza'
kWaterRippleCodecType	= 'ripl'
kWindowsRawCodecType	= 'WRAW'
kYUV420CodecType	= 'y420'

'avr ', 'gif ',...

Note that the fourth character of these type values is a space (ASCII 32).

kComponentVideoSigned

For historical reasons, 'yuvu' identifies the signed type.

kComponentVideoUnsigned

For historical reasons, 'yuvs' identifies the unsigned type.

Discussion

All codec components of the same type provide the same kinds of services and support a common application programming interface.

See Also

For more general component types, see “Component Types” (IV–2621).

Color Constants

Identify default colors for a graphics importer component.

blackColor	= 33
blueColor	= 409
cyanColor	= 273
greenColor	= 341
magentaColor	= 137
redColor	= 205
whiteColor	= 30
yellowColor	= 69

See Also

For an example of passing color constants, see `GraphicsImportGetDefaultGraphicsMode` (I-630).

Component Identifiers

Identify the types of components.

BaseMediaType	= 'gnrc'
clockComponentType	= 'clock'
compressorComponentType	= 'imco'
CreateFilePreviewComponentType	= 'pmak'
DataHandlerType	= 'dh1r'
decompressorComponentType	= 'imdc'
FlashMediaType	= 'flsh'
HandleDataHandlerSubType	= 'hndl'
MediaHandlerType	= 'mhlr'
MovieControllerComponentType	= 'play'
MovieExportType	= 'spit'
MovieImportType	= 'eat '
MovieMediaType	= 'moov'
MPEGMediaType	= 'MPEG'
MusicMediaType	= 'musi'
ResourceDataHandlerSubType	= 'rsrc'
SeqGrabChannelType	= 'sgch'
SeqGrabComponentType	= 'barg'
SeqGrabCompressionPanelType	= 'cmpr'
SeqGrabPanelType	= 'sgpn'
SeqGrabSourcePanelType	= 'sour'
ShowFilePreviewComponentType	= 'pnot'
SoundMediaType	= 'soun'
SpriteMediaType	= 'sprt'
StandardCompressionSubType	= 'imag'

StandardCompressionSubTypeSound	= 'soun'
StandardCompressionType	= 'scdi'
systemMicrosecondClock	= 'micr'
systemMillisecondClock	= 'mill'
systemSecondClock	= 'seco'
systemTickClock	= 'tick'
TextMediaType	= 'text'
ThreeDeeMediaType	= 'qd3d'
TimeCodeMediaType	= 'tmcd'
TweenMediaType	= 'twen'
URLDataHandlerSubType	= 'url '
videoDigitizerComponentType	= 'vdig'
VideoMediaType	= 'vide'
WiredActionHandlerType	= 'wire'

'eat ', 'url '

Note that the fourth character of these types is a space (ASCII 32).

'imco'

Generic type for components that compress graphic images.

'imdc'

Generic type for components that decompress graphic images.

'micr'

A clock subtype measuring millionths of a second since user startup.

'mill'

A clock subtype measuring thousandths of a second since user startup.

'seco'

A clock subtype measuring seconds since January 1, 1904.

'tick'

A clock subtype measuring sixtieths (1/60) of a second since user startup.

Discussion

All components of the same type or subtype provide the same kinds of services and support a common application programming interface. Codecs have their own set of types.

See Also

For the type constants of codecs, see “Codec Types” (IV–2619). For an example of using component types, see `GetComponentTypeModSeed` (I–395).

Controller Numbers

Identify controllers in the QuickTime Music Architecture.

kControllerModulationWheel	= 1
kControllerBreath	= 2
kControllerFoot	= 4
kControllerPortamentoTime	= 5
kControllerVolume	= 7
kControllerBalance	= 8
kControllerPan	= 10
kControllerExpression	= 11
kControllerLever1	= 16
kControllerLever2	= 17
kControllerLever3	= 18
kControllerLever4	= 19
kControllerLever5	= 80
kControllerLever6	= 81
kControllerLever7	= 82
kControllerLever8	= 83
kControllerPitchBend	= 32
kControllerAfterTouch	= 33
kControllerSustain	= 64
kControllerSostenuto	= 66
kControllerSoftPedal	= 67
kControllerReverb	= 91
kControllerTremolo	= 92
kControllerChorus	= 93
kControllerCeleste	= 94
kControllerPhaser	= 95
kControllerEditPart	= 113
kControllerMasterTune	= 114

kControllerModulationWheel

This controller controls the modulation wheel. A modulation wheel adds a periodic change to the volume or pitch of a sounding tone, depending on the modulation depth knobs.

kControllerBreath

This controller controls breath.

kControllerFoot

This controller controls the foot pedal.

kControllerPortamentoTime

This controller adjusts the slur between notes. Set the time to 0 to turn off portamento; there is no separate control to turn portamento on and off.

kControllerVolume

This controller controls volume.

kControllerBalance

This controller controls balance between channels.

`kControllerPan`

This controller controls balance on the QuickTime music synthesizer and some others. Values are 256–512, corresponding to left to right.

`kControllerExpression`

This controller provides a second volume control.

`kControllerLever1`

Eight general-purpose controllers, `kControllerLever1` through `kControllerLever8`.

`kControllerPitchBend`

This controller bends the pitch. Pitch bend is specified in positive and negative semitones, with 7 bits per fraction.

`kControllerAfterTouch`

This controller controls channel pressure.

`kControllerSustain`

This controller controls the sustain effect. The value is a `Boolean`; positive for on, 0 or negative for off.

`kControllerSostenuto`

This controller controls sostenuto.

`kControllerSoftPedal`

This controller controls the soft pedal.

`kControllerReverb`

This controller controls reverberation.

`kControllerTremolo`

This controller controls tremolo.

`kControllerChorus`

This controller controls the amount of signal to feed to the chorus special effect unit.

`kControllerCeleste`

This controller controls the amount of signal to feed to the celeste special effect unit.

`kControllerPhaser`

This controller controls the amount of signal to feed to the phaser special effect unit.

`kControllerEditPart`

This controller sets the part number for which editing is occurring. For synthesizers that can edit only one part.

kControllerMasterTune

This controller offsets the entire synthesizer in pitch.

Discussion

The controller numbers used by QuickTime are mostly identical to the standard MIDI controller numbers. These are signed 8.8 values. The full range, therefore, is –128.00 to 127+127 / 128 (or 0x8000 to 0x7FFF). All controls default to zero except for volume and pan.

Pitch bend is specified in fractional semitones, which eliminates the restrictions of a pitch bend range. You can bend as far as you want, any time you want.

The last 16 controllers (113 through 128) are global controllers. Global controllers respond when the part number is given as 0, indicating the entire synthesizer.

See Also

See the `SynthesizerDescription` (IV–2465) structure.

Data References

Identify the types of data references.

BaseMediaType	= 'gnrc'
FlashMediaType	= 'flsh'
HandleDataHandlerSubType	= 'hndl'
MovieMediaType	= 'moov'
MPEGMimeType	= 'MPEG'
MusicMediaType	= 'musi'
NullDataHandlerSubType	= 'null'
PointerDataHandlerSubType	= 'ptr '
rAliasType	= 'alis'
ResourceDataHandlerSubType	= 'rsrc'
SoundMediaType	= 'soun'
SpriteMediaType	= 'sprt'
TextMediaType	= 'text'
ThreeDeeMediaType	= 'qd3d'
TimeCodeMediaType	= 'tmcd'
TweenMediaType	= 'twen'
URLDataHandlerSubType	= 'url '
VideoMediaType	= 'vide'
WiredActionHandlerType	= 'wire'

rAliasType

The data reference is a Mac OS **alias**; for information about aliases, see *Inside Macintosh: Macintosh Toolbox Essentials* (listed in the **bibliography**).

See Also

For an example of using data reference type codes, see `DataHGetDataRef` (I–189).

Effects Codes

Identify QuickTime visual effect components.

```
// One-source effects
kBlurImageFilterType           = 'blur'
kSharpenImageFilterType        = 'shrp'
kEdgeDetectImageFilterType     = 'edge'
kEmbossImageFilterType         = 'embs'
kConvolveImageFilterType       = 'genk'
kAlphaGainImageFilterType      = 'gain'
kRGBColorBalanceImageFilterType = 'rgbb'
kHSLColorBalanceImageFilterType = 'hslb'
kColorSyncImageFilterType      = 'sync'
kFilmNoiseImageFilterType      = 'fmns'
kSolarizeImageFilterType       = 'solr'
kColorTintImageFilterType      = 'tint'
kLensFlareImageFilterType      = 'lens'
kBrightnessContrastImageFilterType = 'brco'

// Two-source effects
kAlphaCompositorTransitionType = 'blnd'
kCrossFadeTransitionType       = 'dslv'
kChromaKeyTransitionType       = 'ckey'
kImplodeTransitionType         = 'mplo'
kExplodeTransitionType         = 'xplo'
kGradientTransitionType        = 'matt'
kPushTransitionType            = 'push'
kSlideTransitionType           = 'slid'
kWipeTransitionType            = 'smpt'
kIrisTransitionType            = 'smp2'
kRadialTransitionType          = 'smp3'
kMatrixTransitionType          = 'smp4'
kZoomTransitionType            = 'zoom'
```

Discussion

One-source effects modify a single image. Two-source effects transition between one image and another.

See Also

For an example of using effects codes, see `MakeImageDescriptionForEffect` (II–785).

Error Codes

Identify errors generated while executing QuickTime calls.

// General QuickTime errors	
couldNotResolveDataRef	= -2000
badImageDescription	= -2001
badPublicMovieAtom	= -2002
cantFindHandler	= -2003
cantOpenHandler	= -2004
badComponentType	= -2005
noMediaHandler	= -2006
noDataHandler	= -2007
invalidMedia	= -2008
invalidTrack	= -2009
invalidMovie	= -2010
invalidSampleTable	= -2011
invalidDataRef	= -2012
invalidHandler	= -2013
invalidDuration	= -2014
invalidTime	= -2015
cantPutPublicMovieAtom	= -2016
badEditList	= -2017
mediaTypesDontMatch	= -2018
progressProcAborted	= -2019
movieToolboxUninitialized	= -2020
noRecordOfApp	= -2020
wfFileNotFound	= -2021
cantCreateSingleForkFile	= -2022
invalidEditState	= -2023
nonMatchingEditState	= -2024
staleEditState	= -2025
userDataItemNotFound	= -2026
maxSizeToGrowTooSmall	= -2027
badTrackIndex	= -2028
trackIDNotFound	= -2029
trackNotInMovie	= -2030
timeNotInTrack	= -2031
timeNotInMedia	= -2032
badEditIndex	= -2033
internalQuickTimeError	= -2034
cantEnableTrack	= -2035
invalidRect	= -2036
invalidSampleNum	= -2037
invalidChunkNum	= -2038
invalidSampleDescIndex	= -2039
invalidChunkCache	= -2040
invalidSampleDescription	= -2041
dataNotOpenForRead	= -2042

dataNotOpenForWrite	= -2043
dataAlreadyOpenForWrite	= -2044
dataAlreadyClosed	= -2045
endOfDataReached	= -2046
dataNoDataRef	= -2047
noMovieFound	= -2048
invalidDataRefContainer	= -2049
badDataRefIndex	= -2050
noDefaultDataRef	= -2051
couldNotUseAnExistingSample	= -2052
featureUnsupported	= -2053
unsupportedAuxiliaryImportData	= -2057
auxiliaryExportDataUnavailable	= -2058
samplesAlreadyInMediaErr	= -2059
noSourceTreeFoundErr	= -2060
sourceNotFoundErr	= -2061
movieTextNotFoundErr	= -2062
missingRequiredParameterErr	= -2063
invalidSpriteWorldPropertyErr	= -2064
invalidSpritePropertyErr	= -2065
gWorldsNotSameDepthAndSizeErr	= -2066
invalidSpriteIndexErr	= -2067
invalidImageIndexErr	= -2068
invalidSpriteIDErr	= -2069
// QuickTime Music Architecture errors	
internalComponentErr	= -2070
notImplementedMusicOSErr	= -2071
cantSendToSynthesizerOSErr	= -2072
cantReceiveFromSynthesizerOSErr	= -2073
illegalVoiceAllocationOSErr	= -2074
illegalPartOSErr	= -2075
illegalChannelOSErr	= -2076
illegalKnobOSErr	= -2077
illegalKnobValueOSErr	= -2078
illegalInstrumentOSErr	= -2079
illegalControllerOSErr	= -2080
midiManagerAbsentOSErr	= -2081
synthesizerNotRespondingOSErr	= -2082
synthesizerOSErr	= -2083
illegalNoteChannelOSErr	= -2084
noteChannelNotAllocatedOSErr	= -2085
tunePlayerFullOSErr	= -2086
tuneParseOSErr	= -2087
noExportProcAvailableErr	= -2089
videoOutputInUseErr	= -2090
// Windows-specific errors	
componentDllLoadErr	= -2091

componentDllEntryNotFoundErr	= -2092
qtmlDllLoadErr	= -2093
qtmlDllEntryNotFoundErr	= -2094
qtmlUninitialized	= -2095
unsupportedOSErr	= -2096
unsupportedProcessorErr	= -2097
noVideoTrackInMovieErr	= -2054
noSoundTrackInMovieErr	= -2055
soundSupportNotAvailableErr	= -2056
// QT atom errors	
cannotFindAtomErr	= -2101
notLeafAtomErr	= -2102
atomsNotOfSameTypeErr	= -2103
atomIndexInvalidErr	= -2104
duplicateAtomTypeAndIDErr	= -2105
invalidAtomErr	= -2106
invalidAtomContainerErr	= -2107
invalidAtomTypeErr	= -2108
cannotBeLeafAtomErr	= -2109
// Data access errors	
pathTooLongErr	= -2110
emptyPathErr	= -2111
noPathMappingErr	= -2112
pathNotVerifiedErr	= -2113
unknownFormatErr	= -2114
wackBadFileErr	= -2115
wackForkNotFoundErr	= -2116
wackBadMetaDataErr	= -2117
qfcbNotFoundErr	= -2118
qfcbNotCreatedErr	= -2119
AAPNotCreatedErr	= -2120
AAPNotFoundErr	= -2121
ASDBadHeaderErr	= -2122
ASDBadForkErr	= -2123
ASDEntryNotFoundErr	= -2124
fileOffsetTooBigErr	= -2125
notAllowedToSaveMovieErr	= -2126
qtNetworkAlreadyAllocatedErr	= -2127
urlDataHTTPProtocolErr	= -2129
urlDataHTTPNoNetDriverErr	= -2130
urlDataHTTPURLErr	= -2131
urlDataHTTPRedirectErr	= -2132
urlDataHFTPProtocolErr	= -2133
urlDataHFTPShutdownErr	= -2134
urlDataHFTPBadUserErr	= -2135
urlDataHFTPBadPasswordErr	= -2136
urlDataHFTPServerError	= -2137

urlDataHFTPDataConnectionErr	= -2138
urlDataHFTPNoDirectoryErr	= -2139
urlDataHFTPQuotaErr	= -2140
urlDataHFTPPermissionsErr	= -2141
urlDataHFTPFilenameErr	= -2142
urlDataHFTPNoNetDriverErr	= -2143
urlDataHFTPBadNameListErr	= -2144
urlDataHFTPNeedPasswordErr	= -2145
urlDataHFTPNoPasswordErr	= -2146
urlDataHFTPServerDisconnectedErr	= -2147
urlDataHFTPURLerr	= -2148
notEnoughDataErr	= -2149
qtActionNotHandledErr	= -2157
// Digitizing errors	
digiUnimpErr	= -2201
qtParamErr	= -2202
matrixErr	= -2203
notExactMatrixErr	= -2204
noMoreKeyColorsErr	= -2205
notExactSizeErr	= -2206
badDepthErr	= -2207
noDMAErr	= -2208
badCallOrderErr	= -2209
// Codec errors	
codecErr	= -8960
noCodecErr	= -8961
codecUnimpErr	= -8962
codecSizeErr	= -8963
codecScreenBufErr	= -8964
codecImageBufErr	= -8965
codecSpoolErr	= -8966
codecAbortErr	= -8967
codecWouldOffscreenErr	= -8968
codecBadDataErr	= -8969
codecDataVersErr	= -8970
codecExtensionNotFoundErr	= -8971
scTypeNotFoundErr	= -8971
codecConditionErr	= -8972
codecOpenErr	= -8973
codecCantWhenErr	= -8974
codecCantQueueErr	= -8975
codecNothingToBlitErr	= -8976
codecNoMemoryPleaseWaitErr	= -8977
codecDisabledErr	= -8978
codecNeedToFlushChainErr	= -8979
lockPortBitsBadSurfaceErr	= -8980
lockPortBitsWindowMovedErr	= -8981


```
lockPortBitsWindowResizedErr      = -8982
lockPortBitsWindowClippedErr      = -8983
lockPortBitsBadPortErr            = -8984
lockPortBitsSurfaceLostErr        = -8985
codecParameterDialogConfirm        = -8986
codecNeedAccessKeyErr             = -8987
codecOffscreenFailedErr            = -8988
codecDroppedFrameErr              = -8989
directXObjectAlreadyExists         = -8990
lockPortBitsWrongGDeviceErr        = -8991
codecOffscreenFailedPleaseRetryErr  = -8992

// QuickTime VR Errors
notAQTVMovieErr                   = -30540
constraintReachedErr              = -30541
callNotSupportedByNodeErr         = -30542
selectorNotSupportedByNodeErr     = -30543
invalidNodeIDErr                  = -30544
invalidViewStateErr               = -30545
timeNotInViewErr                  = -30546
propertyNotSupportedByNodeErr     = -30547
settingNotSupportedByNodeErr      = -30548
limitReachedErr                   = -30549
invalidNodeFormatErr              = -30550
invalidHotSpotIDErr               = -30551
noMemoryNodeFailedInitialize      = -30552
streamingNodeNotReadyErr          = -30553
qtvrlibraryLoadErr                = -30554
qtvrUninitialized                  = -30555
```

noRecordOfApp

A replica of the movieToolboxUninitialized error.

cantCreateSingleForkFile

The file to be created already exists.

componentDllLoadErr

Windows error returned when a component is loading.

componentDllEntryNotFoundErr

Windows error returned when a component is loading.

qtmDllLoadErr

Windows error returned when the QuickTime Media Layer is loading.

qtmDllEntryNotFoundErr

Windows error returned when the QuickTime Media Layer is loading.

digiUnimpErr

Digitizer feature is unimplemented.

qtParamErr

Bad input parameter (out of range, for example).

matrixErr

Bad matrix; the digitizer did nothing.

notExactMatrixErr

Warning of a bad matrix; the digitizer did its best.

noMoreKeyColorsErr

All the key indexes are in use.

notExactSizeErr

Can't digitize to the exact size requested.

badDepthErr

Can't digitize into the requested pixel depth.

noDMAErr

Can't do DMA digitizing; that is, can't go to the requested destination.

badCallOrderErr

A status call was made before being set up first.

Discussion

The Movie Toolbox provides two error values to your application: the current error and the **sticky error**. The current error is the result code from the last Movie Toolbox function; it is updated each time your application calls a Movie Toolbox function. The sticky error value contains the first nonzero result code from any Movie Toolbox function that you called after having cleared the sticky error with `ClearMoviesStickyError` (I-90).

See Also

For examples of retrieving error codes, see `GetMoviesError` (I-492) and `GetMoviesStickyError` (I-494).

File Types and Creators

Identify the formats of graphics files and the applications that create them.

```
// File types
ftAdobePremiereMovie    = 'MooV'
ftAfterDarkModule        = 'ADgm'
ftClip3Dgraphic          = 'EZ3D'
ftCricketChart           = 'CGPC'
ftCricketDrawing         = 'CKDT'
ftDesignCADDrawing       = 'DCAD'
```

```

ftImageStudioGraphic      = 'RIFF'
ftKaleidaGraphGraphic     = 'QPCT'
ftMacFlowChart            = 'FLCH'
ftMacSpinDataSet          = 'D2BN'
ftMoviePlayerMovie        = 'MooV'
ftPixelPaint              = 'PX01'
ftSuper3DDrawing          = '3DBX'
ftSwivel3DDrawing         = 'SMDL'
ftVersaCADDrawing         = '2D  '

// Creator codes
sigAdobePremiere          = 'PrMr'
sigAfterDark              = 'ADrk'
sigAldusSuper3D           = 'SP3D'
sigAutoCAD                 = 'ACAD'
sigClip3D                 = 'EZ3E'
sigColorMacCheese         = 'CMCΔ'
sigCricketDraw            = 'CRDW'
sigCricketGraph           = 'CGRF'
sigDeltagraphPro          = 'DGRH'
sigDesign2                = 'DESG'
sigDesignCAD              = 'ASBC'
sigDesignStudio           = 'MRJN'
sigDigDarkroom            = 'DIDR'
sigDreams                 = 'PHNX'
sigDynaperspective        = 'PERS'
sigGenericCADD            = 'CAD3'
sigGraphMaster            = 'GRAM'
sigImageStudio            = 'FSPE'
sigInfiniD                = 'SI∞D'
sigKaleidaGraph           = 'QKPT'
sigKidPix                 = 'Kid2'
sigLabVIEW                = 'LBVW'
sigMacDraft               = 'MD20'
sigMacDraw                = 'MDRW'
sigMacFlow                = 'MCFL'
sigMacSpin                = 'D2SP'
sigMiniCad                = 'CDP3'
sigModelShop              = 'MDSP'
sigMoviePlayer            = 'TVOD'
sigMovieRecorder          = 'mrcr'
sigOasis                  = 'TAOA'
sigOBJECTMASTER           = 'BROW'
sigOfoto                  = 'APLS'
sigOmnis5                 = 'Q2$$'
sigOptix                  = 'PIXL'
sigPhotoMac               = 'PMAC'
sigPictureCompressor      = 'ppxi'
sigPictViewer             = 'MDTS'

```

sigPixelFormat	= 'PIXR'
sigScreenPlay	= 'SPLY'
sigSmoothie	= 'Smoo'
sigStudio1	= 'ST/1'
sigStudio32	= 'ST32'
sigStudio8	= 'ST/8'
sigSwivel3D	= 'SWVL'
sigVersaCad	= 'VCAD'

Discussion

Constant names for creator codes are written as `sig` followed by the application name. Constant names for file types are written as `ft` followed by the document type.

See Also

For an example of using file type and creator codes, see [GraphicsImportExportImageFile \(I-616\)](#).

Graphics Transfer Modes

Determine how images will be transferred.

```
// Boolean modes
// src modes are used with bitmaps and text;
// pat modes are used with lines and shapes
srcCopy          = 0
srcOr             = 1
srcXor            = 2
srcBic            = 3
notSrcCopy       = 4
notSrcOr         = 5
notSrcXor        = 6
notSrcBic        = 7
patCopy          = 8
patOr            = 9
patXor           = 10
patBic           = 11
notPatCopy       = 12
notPatOr         = 13
notPatXor        = 14
notPatBic        = 15

// Text dimming
grayishTextOr    = 49

// Highlighting
hilite           = 50
```

hilitetransfermode = 50

// Arithmetic modes

blend = 32

addPin = 33

addOver = 34

subPin = 35

addMax = 37

adMax = 37

subOver = 38

adMin = 39

ditherCopy = 64

// Transparent mode

transparent = 36

srcCopy, patCopy

If the source is black, apply the foreground color to the destination; if the source is white, apply the background color; otherwise apply weighted portions of the foreground and background colors.

srcOr, patOr

If the source is black, apply the foreground color to the destination; if the source is white, do nothing; otherwise apply weighted portions of the foreground color.

srcXor, patXor

If the source is black, invert the destination (this operation is undefined for a colored destination). Otherwise, do nothing.

srcBic, patBic

If the source is black, apply the background color to the destination. If the source is white, do nothing. Otherwise, apply weighted portions of the background color.

notSrcCopy, notPatCopy

If the source is white, apply the foreground color to the destination; if the source is black, apply the background color; otherwise apply weighted portions of the foreground and background colors.

notSrcOr, notPatOr

If the source is white, apply the foreground color to the destination; if the source is black, do nothing; otherwise apply weighted portions of the foreground color.

notSrcXor, notPatXor

If the source is white, invert the destination (this operation is undefined for a colored destination pixel). Otherwise, do nothing.

notSrcBic, notPatBic

If the source is white, apply the background color to the destination. If the source is black, do nothing. Otherwise, apply weighted portions of the background color.

grayishTextOr

Dim the destination. If in color, replace it with a blend of the foreground and background; if black-and-white, replace it with **dithered** black and white. This mode is used primarily for text.

hilite, hilitetransfermode

Replace the background color with the highlight color.

blend

Replace the destination with a blend of the source and destination colors. If the destination is a bitmap, this is the same as srcCopy.

addPin

Replace the destination with the sum of the source and destination, up to a maximum value. If the destination is a bitmap, this is the same as srcBic.

addOver

Replace the destination with the sum of the source and destination, but if the resulting red, green, or blue value exceeds 65536, then subtract 65536 from it. If the destination is a bitmap, this is the same as srcXor.

subPin

Replace the destination with the difference between the source and destination, but not less than a minimum value. If the destination is a bitmap, this is the same as srcOr.

addMax, adMax

Compare the source and destination, and replace the destination with the greater value of each of the red, green, and blue components. If the destination is a bitmap, this is the same as srcBic.

subOver

Replace the destination with the difference between the source and destination, but if the resulting red, green, or blue value is negative, then add 65536 to it. If the destination is a bitmap, this is the same as srcXor.

adMin

Compare the source and destination, and replace the destination with the lesser value of each of the red, green, and blue components. If the destination is a bitmap, this is the same as srcOr.

ditherCopy

Replace the destination with a **dither** mix of the source and destination.

transparent

Replace the destination with the source if the source is not equal to the background.

See Also

For more information about graphics transfer modes, see *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**).

Localization Codes

Identify languages, scripts, numbering systems, calendar systems, and geographical regions.

```
// Language codes:
langAfrikaans           = 141 // smRoman script
langBreton              = 142 // smRoman or smRoman/Celtic script
langAlbanian            = 36  // smRoman script
langAmharic             = 85  // smEthiopic script
langArabic              = 12  // smArabic script
langArmenian            = 51  // smArmenian script
langAssamese            = 68  // smBengali script
langAymara              = 134 // smRoman script
langAzerbaijanAr       = 50  // Azerbaijani in smArabic script
langAzerbaijani        = 49  // Azerbaijani in smCyrillic script
langBasque              = 129 // smRoman script
langBelorussian        = 46  // Synonym for langByelorussian
langUzbek              = 47  // smCyrillic script
langBengali             = 67  // smBengali script
langBulgarian          = 44  // smCyrillic script
langBurmese             = 77  // smBurmese script
langByelorussian       = 46  // smCyrillic script
langCatalan            = 130 // smRoman script
langChewa              = 92  // synonym for langNyanja
langCroatian           = 18  // modified smRoman/Croatian script
langCzech              = 38  // smCentralEuroRoman script
langDanish             = 7   // smRoman script
langDutch              = 4   // smRoman script
langDzongkha           = 137 // (Bhutan) smTibetan script
```

langEnglish	= 0	// smRoman script
langEsperanto	= 94	// smRoman script
langEstonian	= 27	// smCentralEuroRoman script
langFaroese	= 30	// smRoman/Icelandic script
langFarsi	= 31	// modified smArabic/Farsi script
langFinnish	= 13	// smRoman script
langFlemish	= 34	// smRoman script
langFrench	= 1	// smRoman script
langGalician	= 140	// smRoman script
langGeorgian	= 52	// smGeorgian script
langGerman	= 2	// smRoman script
langGreek	= 14	// Greek script using smRoman script
langGreekPoly	= 148	// smGreek script
langGreenlandic	= 149	// smRoman script
langGuarani	= 133	// smRoman script
langGujarati	= 69	// smGujarati script
langHebrew	= 10	// smHebrew script
langHindi	= 21	// smDevanagari script
langHungarian	= 26	// smCentralEuroRoman script
langIcelandic	= 15	// modified smRoman/Icelandic script
langIndonesian	= 81	// smRoman script
langInuktitut	= 143	// Inuit using smEthiopic script
langIrishGaelic	= 35	// smRoman or smRoman/Celtic script
langIrishGaelicScript	= 146	// smRoman/Gaelic script
langItalian	= 3	// smRoman script
langJapanese	= 11	// smJapanese script
langJavaneseRom	= 138	// Javanese in smRoman script
langKannada	= 73	// smKannada script
langKashmiri	= 61	// smArabic script
langKazakh	= 48	// smCyrillic script
langKhmer	= 78	// smKhmer script
langKinyarwanda	= 90	// smRoman script
langKirghiz	= 54	// smCyrillic script
langKorean	= 23	// smKorean script
langKurdish	= 60	// smArabic script
langLao	= 79	// smLao script
langLatin	= 131	// smRoman script
langLatvian	= 28	// smCentralEuroRoman script
langLithuanian	= 24	// smCentralEuroRoman script
langMacedonian	= 43	// smCyrillic script
langMalagasy	= 93	// smRoman script
langMalayalam	= 72	// smMalayalam script
langMalayArabic	= 84	// Malay in smArabic script
langMalayRoman	= 83	// Malay in smRoman script
langMaltese	= 16	// smRoman script
langManxGaelic	= 145	// smRoman or smRoman/Celtic script
langMarathi	= 66	// smDevanagari script
langMoldavian	= 53	// smCyrillic script
langMongolian	= 57	// Mongolian in smMongolian script


```

langMongolianCyr      = 58      // Mongolian in smCyrillic script
langNepali            = 64      // smDevanagari script
langNorwegian         = 9       // smRoman script
langNyanja            = 92      // smRoman script
langOriya             = 71      // smOriya script
langOromo             = 87      // smEthiopic script
langPashto            = 59      // smArabic script
langPersian           = 31      // Synonym for langFarsi
langPolish            = 25      // smCentralEuroRoman script
langPortuguese        = 8       // smRoman script
langPunjabi           = 70      // smGurmukhi script
langQuechua           = 132     // smRoman script
langRomanian          = 37      // modified smRoman/Romanian script
langRuanda            = 90      // synonym for langKinyarwanda
langRundi             = 91      // smRoman script
langRussian           = 32      // smCyrillic script
langSami              = 29      // language of the Sami in Scandanavia
langSanskrit          = 65      // smDevanagari script
langScottishGaelic    = 144     // smRoman or smRoman/Celtic script
langSerbian           = 42      // smCyrillic script
langSimpChinese       = 33      // Mandarin in smSimpChinese script
langSindhi            = 62      // smArabic script
langSinhalese         = 76      // smSinhalese script
langSlovak            = 39      // smCentralEuroRoman script
langSlovenian         = 40      // modified smRoman/Croatian script
langSomali            = 88      // smRoman script
langSpanish           = 6       // smRoman script
langSundaneseRom      = 139     // Sundanese in smRoman script
langSwahili           = 89      // smRoman script
langSwedish           = 5       // smRoman script
langTagalog           = 82      // smRoman script
langTajiki            = 55      // smCyrillic script
langTamil             = 74      // smTamil script
langTatar             = 135     // smCyrillic script
langTelugu            = 75      // smTelugu script
langThai              = 22      // smThai script
langTibetan           = 63      // smTibetan script
langTigrinya          = 86      // smEthiopic script
langTongan            = 147     // smRoman script
langTradChinese       = 19      // Mandarin in smTradChinese script
langTurkish           = 17      // modified smRoman/Turkish script
langTurkmen           = 56      // smCyrillic script
langUighur            = 136     // smArabic script
langUkrainian         = 45      // modified smCyrillic/Ukrainian script
langUrdu              = 20      // smArabic script
langVietnamese        = 80      // smVietnamese script
langWelsh             = 128     // modified smRoman/Celtic script
langYiddish           = 41      // smHebrew script
langUnspecified       = 32767

```

```

// Script codes
smArabic           = 4
smArmenian         = 24
smBengali           = 13
smBurmese           = 19
smCentralEuroRoman = 29
smCyrillic          = 7
smDevanagari        = 9
smEthiopic          = 28
smExtArabic         = 31    // extended Arabic
smGeez              = 28    // Synonym for smEthiopic
smGeorgian          = 23
smGreek             = 6
smGujarati          = 11
smGurmukhi          = 10
smHebrew            = 5
smJapanese          = 1
smKannada            = 16    // Kannada/Kanarese
smKhmer             = 20    // Khmer/Cambodian
smKorean            = 3
smLao               = 22
smMalayalam         = 17
smMongolian         = 27
smOriya             = 12
smRoman             = 0
smRSymbol           = 8     // Right-left symbol
smSimpChinese       = 25    // Simplified Chinese
smSinhalese         = 18
smTamil             = 14
smTelugu            = 15
smThai              = 21
smTibetan           = 26
smTradChinese       = 2     // Traditional Chinese
smUnicodeScript     = 0x7E  // Unicode
smUninterp          = 32    // Uninterpreted symbols
smVietnamese        = 30

// Calendar codes
calGregorian        = 0
calArabicCivil       = 1
calArabicLunar       = 2
calJapanese          = 3
calJewish            = 4
calCoptic            = 5
calPersian           = 6

// Integer format codes
intWestern           = 0

```

```
intArabic           = 1
intRoman            = 2
intJapanese         = 3
intEuropean         = 4

// Region codes
verAfrikaans        = 102
verArabic           = 16
verArmenian         = 84
verAustralia        = 15
verAustria          = 92
verBengali          = 60
verBhutan           = 83
verBrazil           = 71
verBreton           = 77
verBritain          = 2
verBulgaria         = 72
verByeloRussian     = 61
verCatalonia        = 73
verChina            = 52
verCroatia          = 68
verCyprus            = 23
verCzech            = 56
verDenmark          = 9
verEngCanada        = 82
verEsperanto        = 103
verEstonia          = 44
verFarEastGeneric   = 58
verFaroeIsl         = 47
verFinland          = 17
verFlemish          = 6
verFrance           = 1
verFrBelgium        = 98
verFrCanada         = 11
verFrenchUniversal  = 91
verFrSwiss          = 18
verGeorgian         = 85
verGermany          = 3
verGreece           = 20
verGreecePoly       = 40
verGreenland        = 107
verGrSwiss          = 19
verGujarati         = 94
verHungary          = 43
verIceland          = 21
verIndiaHindi       = 33
verIndiaUrdu        = 96
verInternational    = 37
verIran             = 48
```

verIreland	= 50
verIrishGaelicScript	= 81
verIsrael	= 13
verItalianSwiss	= 36
verItaly	= 4
verJapan	= 14
verKorea	= 51
verLatvia	= 45
verLithuania	= 41
verMacedonian	= 67
verMagyar	= 59
verMalta	= 22
verManxGaelic	= 76
verMarathi	= 104
verMultilingual	= 74
verNepal	= 106
verNetherlands	= 5
verNorway	= 12
verNunavut	= 78
verNynorsk	= 101
verPakistanUrdu	= 34
verPoland	= 42
verPortugal	= 10
verPunjabi	= 95
verRomania	= 39
verRussia	= 49
verSami	= 46
verScottishGaelic	= 75
verScriptGeneric	= 55
verSerbian	= 65
verSingapore	= 100
verSlovak	= 57
verSlovenian	= 66
verSpain	= 8
verSpLatinAmerica	= 86
verSweden	= 7
verTaiwan	= 53
verThailand	= 54
verTibetan	= 105
verTonga	= 88
verTurkey	= 24
verTurkishModified	= 35
verUkraine	= 62
verUS	= 0
verUzbek	= 99
verVietnam	= 97
verWelsh	= 79

`langIrishGaelic, verIreland`
Irish Gaelic for Ireland (without dot above).

`langIrishGaelicScript, verIrishGaelicScript`
Irish Gaelic for Ireland (using dot above).

`langSimpChinese, smSimpChinese, verChina`
Chinese using simplified characters.

`langTradChinese, smTradChinese, verTaiwan`
Chinese using traditional characters.

`smCentralEuroRoman`
Script for Czech, Slovak, Polish, Hungarian, and the Baltic languages.

`smRSymbol`
Right-left symbol for bidirectional scripts (such as Arabic and Hebrew).

`verFarEastGeneric`
Generic Far East system (no language or script).

`verGreece`
Monotonic modern Greek.

`verGreecePoly`
Polytonic ancient Greek.

`verInternational`
English for international use.

`verMultilingual`
No language or script.

`verScriptGeneric`
Generic script system (no language or script).

`verSpain`
Spanish for Spain.

`verSpLatinAmerica`
Spanish for Latin America.

See Also

For more information about localization codes, see *Inside Macintosh: Text* (listed in the **bibliography**). For general information about localization, see *Guide to Macintosh Software Localization* (Addison-Wesley 1992, ISBN 0-201-60856-1).

Movie Controller Actions

Specify commands to a movie controller component.

	Value	Parameter, if any
mcActionActivate	= 3	// no parameter
mcActionAdjustCursor	= 65	// pointer to EventRecord (WindowPtr is in message parameter)
mcActionAutoPlay	= 83	// Fixed, play rate
mcActionBadgeClick	= 44	// pointer to Boolean. (Set to false to ignore click.)
mcActionClickAndHoldPoint	= 67	// point (local coordinates). return true if point has click & hold action (e.g. VR object movie autorotate spot)
mcActionControllerSizeChanged	= 26	// no parameter
mcActionCustomButtonClick	= 60	// pointer to EventRecord
mcActionDeactivate	= 4	// no parameter
mcActionDoScript	= 78	// QTDoScriptPtr
mcActionDraw	= 2	// WindowPtr
mcActionEvaluateExpression	= 73	// QTEvaluateExpressionPtr
mcActionExecuteAllActionsForQTEvent	= 63	// ResolvedQTEventSpecPtr
mcActionExecuteOneActionForQTEvent	= 64	// ResolvedQTEventSpecPtr
mcActionFetchParameterAs	= 74	// QTFetchParameterAsPtr
mcActionForceTimeTableUpdate	= 61	// no parametereter
mcActionGetChapterTime	= 71	// QTGetChapterTimePtr
mcActionGetCursorByID	= 75	// QTGetCursorByIDPtr
mcActionGetCursorSettingEnabled	= 56	// pointer to Boolean
mcActionGetDragEnabled	= 51	// pointer to Boolean
mcActionGetExternalMovie	= 70	// QTGetExternalMoviePtr
mcActionGetFlags	= 39	// pointer to long (see below)
mcActionGetIndChapter	= 80	// QTChapterInfoPtr
mcActionGetKeysEnabled	= 33	// pointer to Boolean
mcActionGetLooping	= 22	// pointer to Boolean
mcActionGetLoopIsPalindrome	= 24	// pointer to Boolean
mcActionGetNextURL	= 76	// Handle to URL
mcActionGetPlayEveryFrame	= 41	// pointer to Boolean
mcActionGetPlayRate	= 42	// pointer to Fixed
mcActionGetPlaySelection	= 35	// pointer to Boolean
mcActionGetSelectionBegin	= 53	// TimeRecord
mcActionGetSelectionDuration	= 54	// TimeRecord
mcActionGetTimeSliderRect	= 49	// pointer to Rect
mcActionGetUseBadge	= 37	// pointer to Boolean
mcActionGetVolume	= 15	// pointer to short integer
mcActionGoToTime	= 12	// TimeRecord
mcActionIdle	= 1	// no parameter
mcActionKey	= 6	// pointer to EventRecord
mcActionLinkToURL	= 59	// Handle to URL
mcActionMouseDown	= 5	// pointer to EventRecord
mcActionMovieChanged	= 77	

```

mcActionMovieClick                = 45 // pointer to event record.
                                   (Change "what" to nullEvt to kill click.)
mcActionMovieEdited                = 50 // no parameter
mcActionPerformActionList         = 72 // QTAtomSpecPtr
mcActionPlay                      = 8  // Fixed play rate
mcActionPrerollAndPlay            = 55 // Fixed play rate
mcActionRestartAtTime            = 79 // QTResartAtTimePtr
mcActionResume                   = 47 // no parameter
mcActionSetColorTable            = 58 // CTabHandle
mcActionSetControllerKeysEnabled = 48 // Boolean
mcActionSetControllerTimeLimits   = 62 // pointer to 2 time values
                                   min/max. Do not send this message to controller. use internally only.
mcActionSetCursorSettingEnabled   = 57 // Boolean
mcActionSetDragEnabled            = 52 // Boolean
mcActionSetFlags                  = 38 // pointer to long (see below)
mcActionSetGrowBoxBounds          = 25 // Rect
mcActionSetKeysEnabled            = 32 // Boolean
mcActionSetLooping                = 21 // Boolean
mcActionSetLoopIsPalindrome       = 23 // Boolean
mcActionSetPlayEveryFrame         = 40 // Boolean
mcActionSetPlaySelection          = 34 // Boolean
mcActionSetSelectionBegin         = 29 // TimeRecord
mcActionSetSelectionDuration       = 30 // TimeRecord;
                                   Action only taken on set-duration.

mcActionSetUseBadge               = 36 // Boolean
mcActionSetVolume                 = 14 // short integer
mcActionShowBalloon               = 43 // pointer to Boolean.
                                   (set to false to stop balloon.)

mcActionShowMessageString         = 68 // StringPtr
mcActionShowStatusString          = 69 // QTStatusStringPtr
mcActionStep                      = 18 // number of steps (short)
mcActionSuspend                   = 46 // no parameter
mcActionUseTrackForTimeTable      = 66 // pointer to {long trackID;
                                   Boolean useIt}. Do not send this message to controller.

```

mcActionGetFlags Constants

mcFlagSuppressMovieFrame

If this flag is set to 1, the controller does not display a frame around the movie.
By default, this flag is set to 0.

mcFlagSuppressStepButtons

If this flag is set to 1, the controller does not display the step buttons. By default,
this flag is set to 0.

mcFlagSuppressSpeakerButton

If this flag is set to 1, the controller does not display the speaker button. By
default, this flag is set to 0.

`mcFlagsUseWindowPalette`

If this flag is set to 1, the movie controller does not manage the window palette. This ensures that a movie's colors are reproduced as accurately as possible. This flag is particularly useful for movies with custom color tables. By default, this flag is set to 0.

mcActionSetFlags Constants

`mcFlagSuppressMovieFrame`

If this flag is set to 1, the controller does not display a frame around the movie. By default, this flag is set to 0.

`mcFlagSuppressStepButtons`

If this flag is set to 1, the controller does not display the step buttons. By default, this flag is set to 0.

`mcFlagSuppressSpeakerButton`

If this flag is set to 1, the controller does not display the speaker button. By default, this flag is set to 0.

See Also

See `MCDoAction` (II-801).

Music Controllers

Specify music controllers for the QuickTime Music Architecture.

<code>kControllerModulationWheel</code>	= 1
<code>kControllerBreath</code>	= 2
<code>kControllerFoot</code>	= 4
<code>kControllerPortamentoTime</code>	= 5
<code>kControllerVolume</code>	= 7
<code>kControllerBalance</code>	= 8
<code>kControllerPan</code>	= 10
<code>kControllerExpression</code>	= 11
<code>kControllerLever1</code>	= 16
<code>kControllerLever2</code>	= 17
<code>kControllerLever3</code>	= 18
<code>kControllerLever4</code>	= 19
<code>kControllerLever5</code>	= 80
<code>kControllerLever6</code>	= 81
<code>kControllerLever7</code>	= 82
<code>kControllerLever8</code>	= 83
<code>kControllerPitchBend</code>	= 32
<code>kControllerAfterTouch</code>	= 33
<code>kControllerSustain</code>	= 64
<code>kControllerSostenuto</code>	= 66

<code>kControllerSoftPedal</code>	= 67
<code>kControllerReverb</code>	= 91
<code>kControllerTremolo</code>	= 92
<code>kControllerChorus</code>	= 93
<code>kControllerCeleste</code>	= 94
<code>kControllerPhaser</code>	= 95
<code>kControllerEditPart</code>	= 113
<code>kControllerMasterTune</code>	= 114

`kControllerModulationWheel`

Controls the modulation wheel. A modulation wheel adds a periodic change to the volume or pitch of a sounding tone, depending on the modulation depth knobs.

`kControllerBreath`

Controls breath.

`kControllerFoot`

Controls the foot pedal.

`kControllerPortamentoTime`

Adjusts the slur between notes. Set the time to 0 to turn off portamento; there is no separate control to turn portamento on and off.

`kControllerVolume`

Controls volume.

`kControllerBalance`

Controls balance between channels.

`kControllerPan`

Controls balance on the QuickTime music synthesizer and some others. Values are 256 to 512, corresponding to left to right.

`kControllerExpression`

Provides a second volume control.

`kControllerLever1`

The `kControllerLever1` through `kControllerLever8` constants identify general-purpose controllers.

`kControllerPitchBend`

Bends the pitch. Pitch bend is specified in positive and negative semitones, with 7 bits per fraction.

`kControllerAfterTouch`

Controls channel pressure.

`kControllerSustain`

Controls the sustain effect. The value is a `Boolean`; positive for on, 0 or negative for off.

`kControllerSostenuto`

Controls *sostenuto*.

`kControllerSoftPedal`

Controls the soft pedal.

`kControllerReverb`

Controls reverberation.

`kControllerTremolo`

Controls tremolo.

`kControllerChorus`

Controls the amount of signal to feed to the chorus special effect unit.

`kControllerCeleste`

Controls the amount of signal to feed to the celeste special effect unit.

`kControllerPhaser`

Controls the amount of signal to feed to the phaser special effect unit.

`kControllerEditPart`

Sets the part number for which editing is occurring, for synthesizers that can edit only one part.

`kControllerMasterTune`

Offsets the pitch of the entire synthesizer.

Discussion

The controller numbers used by QuickTime are mostly identical to the standard MIDI controller numbers. These are signed 8.8 values. The full range, therefore, is -128.00 to 127+127/128 (or 0x8000 to 0x7FFF). All controls default to zero except for volume and pan.

Pitch bend is specified in fractional semitones, which eliminates the restrictions of a pitch bend range. You can bend as far as you want, any time you want.

The last 16 controllers (113 through 128) are global controllers. Global controllers respond when the part number is given as 0, indicating the entire synthesizer.

See Also

See `MusicGetPartController` (II-1001).

Parameter Behaviors

Specify the user interface behavior of a parameter.

kParameterItemEditText	= 'edit'
kParameterItemEditLong	= 'long'
kParameterItemEditFixed	= 'fixd'
kParameterItemEditDouble	= 'doub'
kParameterItemPopUp	= 'popu'
kParameterItemRadioCluster	= 'radi'
kParameterItemCheckBox	= 'chex'
kParameterItemControl	= 'cntl'
kParameterItemLine	= 'line'
kParameterItemColorPicker	= 'pick'
kParameterItemGroupDivider	= 'divi'
kParameterItemStaticText	= 'stat'
kParameterItemDragImage	= 'imag'

kParameterItemEditText

Edit box for text.

kParameterItemEditLong

Edit box for long numbers.

kParameterItemEditFixed

Edit box for fixed point numbers.

kParameterItemEditDouble

Edit box for double numbers.

kParameterItemPopUp

Pop-up menu value for enumerated types.

kParameterItemRadioCluster

Radio button cluster for enumerated types.

kParameterItemCheckBox

Check box for Boolean types.

kParameterItemControl

Item controlled by a standard control of some type.

kParameterItemLine

Line user item.

kParameterItemColorPicker

Color swatch & picker.

kParameterItemGroupDivider

Start of a new group of items.

kParameterItemStaticText

Display “parameter name” as static text.

kParameterItemDragImage

Allow image display, along with drag and drop.

See Also

See `ParameterDataBehavior` (IV–2328).

Pixel Formats

Defines pixel arrangements in images.

k16LE555PixelFormat	= 'L555'	// 16 bit LE RGB 555 (PC)
k16LE5551PixelFormat	= '5551'	// 16 bit LE RGB 5551
k16BE565PixelFormat	= 'B565'	// 16 bit BE RGB 565
k16LE565PixelFormat	= 'L565'	// 16 bit LE RGB 565
k24BGRPixelFormat	= '24BG'	// 24 bit BGR
k32BGRAPixelFormat	= 'BGR'	// 32 bit BGR (Matrox)
k32ABGRPixelFormat	= 'ABGR'	// 32 bit ABGR
k32RGBAPixelFormat	= 'RGBA'	// 32 bit RGBA
kYUVSPixelFormat	= 'yuvs'	// YUV 4:2:2 byte ordering 16-unsigned
kYUVUPixelFormat	= 'yuvu'	// YUV 4:2:2 byte ordering 16-signed
kYVU9PixelFormat	= 'YVU9'	// YVU9 Planar 9
kYUV411PixelFormat	= 'Y411'	// YUV 4:1:1 Interleaved 16
kYVYU422PixelFormat	= 'YVYU'	// YVYU 4:2:2 byte ordering 16
kUYVY422PixelFormat	= 'UYVY'	// UYVY 4:2:2 byte ordering 16
kYUV211PixelFormat	= 'Y211'	// YUV 2:1:1 Packed 8

See Also

See the `PixelFormat` (IV–2332) structure.

QTMA Events

Designate event subtypes for the QuickTime Music Architecture.

kGeneralEventNoteRequest	= 1
kGeneralEventPartKey	= 4
kGeneralEventTuneDifference	= 5
kGeneralEventAtomicInstrument	= 6
kGeneralEventKnob	= 7
kGeneralEventMIDIChannel	= 8
kGeneralEventPartChange	= 9
kGeneralEventNoOp	= 10
kGeneralEventUsedNotes	= 11

kGeneralEventPartMix	= 12
kMarkerEventEnd	= 0
kMarkerEventBeat	= 1
kMarkerEventTempo	= 2

kGeneralEventNoteRequest

Encapsulates a NoteRequest (IV-2323) structure.

kGeneralEventTuneDifference

Contains a standard sequence, with end marker, for the tune difference of a sequence piece.

kGeneralEventAtomicInstrument

Encapsulates an AtomicInstrument record.

kGeneralEventKnob

Pairs of knobID/knobValue values; smallest event is 4 long integers.

kGeneralEventMIDIChannel

Used in tune header; one long word identifies the MIDI channel it originally came from.

kGeneralEventPartChange

Used in tune sequence; one long word identifies the tune part that can now take over this part's note channel, similar to a program change.

kGeneralEventNoOp

Guaranteed to do nothing and be ignored.

kGeneralEventUsedNotes

Four long words specifying which MIDI notes are actually used; formatted 0..127 MSB to LSB.

kGeneralEventPartMix

Three long words: Fixed volume, long balance, long flags.

kMarkerEventEnd

Value of 0 means stop, value other than 0 means ignore.

kMarkerEventBeat

Value 0 is a single beat; anything else is 65536ths-of-a-beat (quarter note).

kMarkerEventTempo

Value same as beat marker, but indicates that a tempo event should be computed (based on where the next beat or tempo marker is) and emitted upon export.

Version Notes

Events of type kGeneralEventTuneDifference, kGeneralEventPartChange, and kGeneralEventNoOp halt QuickTime 2.0 music playing.

See Also

See the TuneSetHeader (III-1967) function.

QTVR Cursors

Define cursors that may appear in QuickTime Virtual Reality.

- 17000 Pointer interface cursor.
- 17483 Mouse down, cursor top right of joystick center.
- 17484 Mouse down, cursor top left of joystick center.
- 17485 Mouse down, cursor top of joystick center.
- 17486 Mouse down, cursor bottom right of joystick center.
- 17487 Mouse down, cursor bottom left of joystick center.
- 17488 Mouse down, cursor bottom of joystick center.
- 17489 Mouse down, cursor right of joystick center.
- 17490 Mouse down, cursor left of joystick center.
- 17491 Mouse down, joystick center cursor.
- 17492 Mouse up, cursor top right of joystick center.
- 17493 Mouse up, cursor top left of joystick center.
- 17494 Mouse up, cursor top of joystick center.
- 17495 Mouse up, cursor bottom right of joystick center.
- 17496 Mouse up, cursor bottom left of joystick center.
- 17497 Mouse up, cursor bottom of joystick center.
- 17498 Mouse up, cursor right of joystick center.
- 17499 Mouse up, cursor left of joystick center.
- 17500 Mouse up, joystick center cursor.
- 17961 Inside right border spinning cursor, both mouse up and mouse down, when tilt angle is between -90° and -78°; object cannot turn right.
- 17962 Inside left border spinning cursor, both mouse up and mouse down, when tilt angle is between -90° and -78°; object cannot turn left.
- 17963 Inside right border spinning cursor, both mouse up and mouse down, when tilt angle is between -78° and -56°; object cannot turn right.
- 17964 Inside left border spinning cursor, both mouse up and mouse down, when tilt angle is between -78° and -56°; object cannot turn left.
- 17965 Inside right border spinning cursor, both mouse up and mouse down, when tilt angle is between -56° and -34°; object cannot turn right.
- 17966 Inside left border spinning cursor, both mouse up and mouse down, when tilt angle is between -56° and -34°; object cannot turn left.
- 17967 Inside right border spinning cursor, both mouse up and mouse down, when tilt angle is between -34° and -11°; object cannot turn right.
- 17968 Inside left border spinning cursor, both mouse up and mouse down, when tilt angle is between -34° and -11°; object cannot turn left.
- 17969 Inside right border spinning cursor, both mouse up and mouse down, when tilt angle is between -11° and 11°; object cannot turn right.
- 17970 Inside left border spinning cursor, both mouse up and mouse down, when tilt angle is between -11° and 11°; object cannot turn left.
- 17971 Inside right border spinning cursor, both mouse up and mouse down,

- 17972 when tilt angle is between 11° and 34°; object cannot turn right.
Inside left border spinning cursor, both mouse up and mouse down,
when tilt angle is between 11° and 34°; object cannot turn left.
- 17973 Inside right border spinning cursor, both mouse up and mouse down,
when tilt angle is between 34° and 56°; object cannot turn right.
- 17974 Inside left border spinning cursor, both mouse up and mouse down,
when tilt angle is between 34° and 56°; object cannot turn left.
- 17975 Inside right border spinning cursor, both mouse up and mouse down,
when tilt angle is between 56° and 78°; object cannot turn right.
- 17976 Inside left border spinning cursor, both mouse up and mouse down,
when tilt angle is between 56° and 78°; object cannot turn left.
- 17977 Inside right border spinning cursor, both mouse up and mouse down,
when tilt angle is between 78° and 90°; object cannot turn right.
- 17978 Inside left border spinning cursor, both mouse up and mouse down,
when tilt angle is between 78° and 90°; object cannot turn left.
- 17979 Inside bottom border spinning cursor, both mouse up and mouse down;
object cannot turn down.
- 17980 Inside top border spinning cursor, both mouse up and mouse down;
object cannot turn up.
- 17981 Inside right border spinning cursor, both mouse up and mouse down,
when tilt angle is between -90° and -78°.
- 17982 Inside left border spinning cursor, both mouse up and mouse down,
when tilt angle is between -90° and -78°.
- 17983 Inside right border spinning cursor, both mouse up and mouse down,
when tilt angle is between -78° and -56°.
- 17984 Inside left border spinning cursor, both mouse up and mouse down,
when tilt angle is between -78° and -56°.
- 17985 Inside right border spinning cursor, both mouse up and mouse down,
when tilt angle is between -56° and -34°.
- 17986 Inside left border spinning cursor, both mouse up and mouse down,
when tilt angle is between -56° and -34°.
- 17987 Inside right border spinning cursor, both mouse up and mouse down,
when tilt angle is between -34° and -11°.
- 17988 Inside left border spinning cursor, both mouse up and mouse down,
when tilt angle is between -34° and -11°.
- 17989 Inside right border spinning cursor, both mouse up and mouse down,
when tilt angle is between -11° and 11°.
- 17990 Inside left border spinning cursor, both mouse up and mouse down,
when tilt angle is between -11° and 11°.
- 17991 Inside right border spinning cursor, both mouse up and mouse down,
when tilt angle is between 11° and 34°.
- 17992 Inside left border spinning cursor, both mouse up and mouse down,
when tilt angle is between 11° and 34°.
- 17993 Inside right border spinning cursor, both mouse up and mouse down,
when tilt angle is between 34° and 56°.
- 17994 Inside left border spinning cursor, both mouse up and mouse down,
when tilt angle is between 34° and 56°.
- 17995 Inside right border spinning cursor, both mouse up and mouse down,
when tilt angle is between 56° and 78°.

- 17996 Inside left border spinning cursor, both mouse up and mouse down, when tilt angle is between 56° and 78°.
- 17997 Inside right border spinning cursor, both mouse up and mouse down, when tilt angle is between 78° and 90°.
- 17998 Inside left border spinning cursor, both mouse up and mouse down, when tilt angle is between 78° and 90°.
- 17999 Inside bottom border spinning cursor, both mouse up and mouse down.
- 18000 Inside top border spinning cursor, both mouse up and mouse down.
- 18499 Mouse-down cursor when over the central part of the movie.
- 18500 Mouse-up cursor when over the central part of the movie.
- 18975 Pan up-right cursor, with both panning up and panning right constrained.
- 18976 Pan up-right cursor, with panning right constrained.
- 18977 Pan up-right cursor, with panning up constrained.
- 18978 Pan up-right cursor.
- 18979 Pan up-left cursor, with both panning up and panning left constrained.
- 18980 Pan up-left cursor, with panning left constrained.
- 18981 Pan up-left cursor, with panning up constrained.
- 18982 Pan up-left cursor.
- 18983 Pan up cursor, with panning constrained.
- 18984 Pan up cursor.
- 18985 Pan down-right cursor, with both panning down and panning right constrained.
- 18986 Pan down-right cursor, with panning right constrained.
- 18987 Pan down-right cursor, with panning down constrained.
- 18988 Pan down-right cursor.
- 18989 Pan down-left cursor, with both panning down and panning left constrained.
- 18990 Pan down-left cursor, with panning left constrained.
- 18991 Pan down-left cursor, with panning down constrained.
- 18992 Pan down-left cursor.
- 18993 Pan down cursor, with panning constrained.
- 18994 Pan down cursor.
- 18995 Pan right cursor, with panning constrained.
- 18996 Pan right cursor.
- 18997 Pan left cursor, with panning constrained.
- 18998 Pan left cursor.
- 18999 Mouse centered, waiting to pan or tilt cursor.
- 19000 Panning interface mouse-up cursor.
- 31494 Translate on, but unable to translate cursor.
- 31495 Translate on cursor.
- 31496 Zooming out cursor, with zooming constrained.
- 31497 Zooming in cursor, with zooming constrained.
- 31498 Conflicting zoom keys cursor.
- 31499 Zooming out cursor.
- 31500 Zooming in cursor.
- 32000 Mouse over link ('link') hot spot cursor.
- 32192 Mouse-up cursor while URL ('url ') hot spot is executing

- 32193 Mouse down on URL ('url ') hot spot cursor
- 32194 Mouse over URL ('url ') hot spot cursor
- 32195 Mouse-up cursor while undefined object hot spot (any hot spot of a type not covered by other hot spot cursors) is executing.
- 32196 Mouse down on undefined object, that is, any hot spot of a type not covered by other hot spot cursors.
- 32197 Mouse over undefined object, that is, any hot spot of a type not covered by other hot spot cursors.
- 32198 Mouse-up cursor while link ('link') hot spot is executing.
- 32199 Mouse down on link ('link') hot spot cursor.

See Also See QTVRCursorRecord (IV–2395). For pictures of the cursors, see Appendix A of *Programming with QuickTime VR* (listed in the **bibliography**).

Sound Commands

Commands for SndDoCommand (III–1831) and SndDoImmediate (III–1832).

- | | | |
|------------------------|-------|--------------------------------------|
| nullCmd | = 0 | |
| quietCmd | = 3 | |
| flushCmd | = 4 | |
| reInitCmd | = 5 | |
| waitCmd | = 10 | |
| pauseCmd | = 11 | |
| resumeCmd | = 12 | |
| callBackCmd | = 13 | |
| syncCmd | = 14 | |
| availableCmd | = 24 | |
| versionCmd | = 25 | |
| volumeCmd | = 46, | // sound manager 3.0 or later only |
| getVolumeCmd | = 47, | // sound manager 3.0 or later only |
| clockComponentCmd | = 50, | // sound manager 3.2.1 or later only |
| getClockComponentCmd | = 51, | // sound manager 3.2.1 or later only |
| scheduledSoundCmd | = 52, | // sound manager 3.3 or later only |
| linkSoundComponentsCmd | = 53, | // sound manager 3.3 or later only |
| soundCmd | = 80 | |
| bufferCmd | = 81 | |
| rateMultiplierCmd | = 86 | |
| getRateMultiplierCmd | = 87 | |

See Also See the SndCommand (IV–2441) structure and the SndDoCommand (III–1831) and SndDoImmediate (III–1832) functions.

Sound Device Features

Indicate the operating features of a sound device.

k8BitRawIn	= (1 << 0)
k8BitTwosIn	= (1 << 1)
k16BitIn	= (1 << 2)
kStereoIn	= (1 << 3)
k8BitRawOut	= (1 << 8)
k8BitTwosOut	= (1 << 9)
k16BitOut	= (1 << 10)
kStereoOut	= (1 << 11)
kReverse	= (1L << 16)
kRateConvert	= (1L << 17)
kCreateSoundSource	= (1L << 18)
kVMAwareness	= (1L << 21)
kHighQuality	= (1L << 22)
kNonRealTime	= (1L << 23)

See Also

See `AudioInfo` (IV–2160).

Sound Formats

Specify the formats of digitized sound.

kSoundNotCompressed	= 'NONE')
k8BitOffsetBinaryFormat	= 'raw ')
k16BitBigEndianFormat	= 'twos')
k16BitLittleEndianFormat	= 'sowt')
kFloat32Format	= 'fl32')
kFloat64Format	= 'fl64')
k24BitFormat	= 'in24')
k32BitFormat	= 'in32')
kMACE3Compression	= 'MAC3')
kMACE6Compression	= 'MAC6')
kCDXA4Compression	= 'cdx4')
kCDXA2Compression	= 'cdx2')
kIMACompression	= 'ima4')
kULawCompression	= 'ulaw')
kALawCompression	= 'alaw')
kMicrosoftADPCMFormat	= 0x6D730002
kDVIIntelIMAFormat	= 0x6D730011
kDVAudioFormat	= 'dvca')
kQDesignCompression	= 'QDMC')
kQUALCOMMCompression	= 'Qc1p')

kOffsetBinary	= k8BitOffsetBinaryFormat
kTwosComplement	= k16BitBigEndianFormat
kLittleEndianFormat	= k16BitLittleEndianFormat
kMPGELayer3Format	= 0x6D730055

See Also

See CmpSoundHeader (IV–2176).

Sound Information Selectors

Specify information to be returned about the user's sound configuration.

siActiveChannels	= 'chac'	// active channels
siActiveLevels	= 'lmac'	// active meter levels
siAGCOnOff	= 'agc '	// automatic gain control state
siAsync	= 'asyn'	// asynchronous capability
siAVDisplayBehavior	= 'avdb'	
siChannelAvailable	= 'chav'	// number of channels available
siCompressionAvailable	= 'cmav'	// compression types available
siCompressionChannels	= 'cpct'	// compressor's number of channels
siCompressionFactor	= 'cmfa'	// current compression factor
siCompressionHeader	= 'cmhd'	// return compression header
siCompressionNames	= 'cnam'	// compression type names avail
siCompressionParams	= 'evaw'	// compression parameters
siCompressionSampleRate	= 'cprt'	// compressor's sample rate
siCompressionType	= 'comp'	// current compression type
siContinuous	= 'cont'	// continous recording
siDecompressionParams	= 'wave'	// decompression parameters
siDeviceBufferInfo	= 'dbin'	// size of interrupt buffer
siDeviceConnected	= 'dcon'	// input device conn status
siDeviceIcon	= 'icon'	// input device icon
siDeviceName	= 'name'	// input device name
siEQSpectrumBands	= 'eqsb'	// number of spectrum bands
siEQSpectrumLevels	= 'eqlv'	// get spectrum meter levels
siEQSpectrumOnOff	= 'eqlo'	// turn on/off spect mtr levels
siEQToneControlGain	= 'eqtg'	// set the bass and treble gain
siEQToneControlOnOff	= 'eqtc'	// turn on equalizer attenuation
siHardwareBalance	= 'hbal'	// balance setting of hardware
siHardwareBalanceSteps	= 'hb1s'	// num balance steps for hardware
siHardwareBass	= 'hbas'	// bass level of hardware
siHardwareBassSteps	= 'hbst'	// num bass steps for hardware
siHardwareBusy	= 'hwbs'	// sound hardware is in use
siHardwareFormat	= 'hwfm'	// get hardware format
siHardwareMute	= 'hmut'	// mute state of all hardware
siHardwareTreble	= 'htrb'	// treble level of hardware
siHardwareTrebleSteps	= 'hwts'	// num treble steps for hardware
siHardwareVolume	= 'hvol'	// volume level of hardware

CON

Constants

siHardwareVolumeSteps	= 'hstp'	// num volume steps for hardware
siHeadphoneMute	= 'pmut'	// mute state of headphones
siHeadphoneVolume	= 'pvol'	// volume level of headphones
siHeadphoneVolumeSteps	= 'hdst'	// num volume steps for headphones
siInputAvailable	= 'inav'	// input sources available
siInputGain	= 'gain'	// input gain
siInputSource	= 'sour'	// input source selector
siInputSourceNames	= 'snam'	// input source names
siLevelMeterOnOff	= 'lmet'	// level meter state
siModemGain	= 'mgai'	// modem input gain
siMonitorAvailable	= 'mnav'	
siMonitorSource	= 'mons'	
siNumberChannels	= 'chan'	// current number of channels
siOptionsDialog	= 'optd'	// display options dialog
siOSTypeInputSource	= 'inpt'	// input source by OSType
siOSTypeInputAvailable	= 'inav'	// list of input source OSTypes
siPlayThruOnOff	= 'plth'	// playthrough state
siPostMixerSoundComponent	= 'psmx'	// install post-mixer effect
siPreMixerSoundComponent	= 'prmx'	// install pre-mixer effect
siQuality	= 'qual'	// quality setting
siRateMultiplier	= 'rmul'	// throttle rate setting
siRecordingQuality	= 'qual'	// recording quality
siSampleRate	= 'srat'	// current sample rate
siSampleRateAvailable	= 'srav'	// sample rates available
siSampleSize	= 'ssiz'	// current sample size
siSampleSizeAvailable	= 'ssav'	// sample sizes available
siSetupCDAudio	= 'sucd'	// set up sound hardware for CD
siSetupModemAudio	= 'sumd'	// set up sound hardware for modem
siSlopeAndIntercept	= 'flap'	// float-point vars for conversion
siSoundClock	= 'sclk'	
siUseThisSoundClock	= 'sclc'	// tell mixer to use sound clock
siSpeakerMute	= 'smut'	// mute state of built-in speaker
siSpeakerVolume	= 'svol'	// volume of built-in speaker
siSSpCPULoadLimit	= '3dll'	
siSSpLocalization	= '3dif'	
siSSpSpeakerSetup	= '3dst'	
siStereoInputGain	= 'sgai'	// stereo input gain
siSubwooferMute	= 'bmut'	// mute state of sub-woofer
siTwosComplementOnOff	= 'twos'	// two's complement state
siVolume	= 'volu'	// volume level of source
siVoxRecordInfo	= 'voxr'	// VOX record parameters
siVoxStopInfo	= 'voxs'	// VOX stop parameters
siWideStereo	= 'wide'	// wide stereo setting

See Also

For an example of using sound information selectors, see `GetSoundOutputInfo` (I-511).

Sound Sample Rates

Encode the rates at which sound data is sampled.

```
rate48khz      = (long)0xBB800000,      // 48000.00000 in fixed-point
rate44khz      = (long)0xAC440000,      // 44100.00000 in fixed-point
rate32khz      = 0x7D000000,           // 32000.00000 in fixed-point
rate22050hz    = 0x56220000,           // 22050.00000 in fixed-point
rate22khz      = 0x56EE8BA3,           // 22254.54545 in fixed-point
rate16khz      = 0x3E800000,           // 16000.00000 in fixed-point
rate11khz      = 0x2B7745D1,           // 11127.27273 in fixed-point
rate11025hz    = 0x2B110000,           // 11025.00000 in fixed-point
rate8khz       = 0x1F400000            // 8000.00000 in fixed-point
```

Discussion

Sample rate constants are declared as a Fixed data type, but the most significant bit is not treated as a sign bit; instead, it is interpreted as having the value 32,768.

See Also

See CmpSoundHeader (IV–2176).

Streaming Transport Atoms

Identify transport atom types for QuickTime streaming.

```
kQTSTransAndProxyAtomType  = 'strp'    // transport/proxy prefs root atom
kQTSConnectionPrefsVersion = 'vers'    // prefs format version
kQTSTransportPrefsAtomType = 'trns'    // tranport prefs root atom
kQTSConnectionAtomType     = 'conn'    // connection prefs atom type
kQTSUDPTransportType       = 'udp '    // udp transport prefs
kQTSHTTPTransportType      = 'http'    // http transport prefs
kQTSTCPTTransportType      = 'tcp '    // tcp transport prefs
kQTSPProxyPrefsAtomType    = 'prxy'    // proxy prefs root atom
kQTSHTTPProxyPrefsType     = 'http'    // http proxy settings
kQTSRTSPProxyPrefsType     = 'rtsp'    // rtsp proxy settings
kQTSSOCKSProxyPrefsType    = 'scks'    // socks proxy settings
kQTSDontProxyPrefsAtomType = 'nopr'    // no-proxy prefs root atom
kQTSDontProxyDataType      = 'data'    // no proxy settings
```

See Also

See QTSTransportPref (IV–2388).

User Data Identifiers

Identify user data fields in a movie.

kUserDataMovieControllerType	= 'ctyp'
kUserDataName	= 'name'
kUserDataTextAuthor	= '@aut'
kUserDataTextComment	= '@cmt'
kUserDataTextCopyright	= '@cpy'
kUserDataTextCreationDate	= '@day'
kUserDataTextDescription	= '@des'
kUserDataTextDirector	= '@dir'
kUserDataTextDisclaimer	= '@dis'
kUserDataTextFullName	= '@nam'
kUserDataTextHostComputer	= '@hst'
kUserDataTextInformation	= '@inf'
kUserDataTextKeywords	= '@key'
kUserDataTextMake	= '@mak'
kUserDataTextModel	= '@mod'
kUserDataTextOriginalFormat	= '@fmt'
kUserDataTextOriginalSource	= '@src'
kUserDataTextPerformers	= '@prf'
kUserDataTextProducer	= '@prd'
kUserDataTextProduct	= '@PRD'
kUserDataTextSoftware	= '@swr'
kUserDataTextSpecialPlaybackRequirements	= '@req'
kUserDataTextWarning	= '@wrn'
kUserDataTextWriter	= '@wrt'
kUserDataTextEditDate1	= '@ed1'
kUserDataTextChapter	= '@chp'

See Also

For an example of passing user data types, see `GetUserData` (I-547).

Video Digitizer Capabilities

Flags that indicate the input and output capabilities of a video digitizer.

// Input capabilities	
digiInDoesNTSC	= 1L<<0
digiInDoesPAL	= 1L<<1
digiInDoesSECAM	= 1L<<2
digiInDoesGenLock	= 1L<<7
digiInDoesComposite	= 1L<<8
digiInDoesSVideo	= 1L<<9
digiInDoesComponent	= 1L<<10

```

digiInVTR_Broadcast      =1L<<11
digiInDoesColor          =1L<<12
digiInDoesBW             =1L<<13
digiInSignalLock         =1L<<31

```

```

// Output capabilities
digiOutDoes1             =1L<<0
digiOutDoes2             =1L<<1
digiOutDoes4             =1L<<2
digiOutDoes8             =1L<<3
digiOutDoes16            =1L<<4
digiOutDoes32            =1L<<5
digiOutDoesDither        =1L<<6
digiOutDoesStretch       =1L<<7
digiOutDoesShrink        =1L<<8
digiOutDoesMask          =1L<<9
digiOutDoesDouble        =1L<<11
digiOutDoesQuad          =1L<<12
digiOutDoesQuarter       =1L<<13
digiOutDoesSixteenth     =1L<<14
digiOutDoesRotate        =1L<<15
digiOutDoesHorizFlip     =1L<<16
digiOutDoesVertFlip      =1L<<17
digiOutDoesSkew          =1L<<18
digiOutDoesBlend         =1L<<19
digiOutDoesWarp          =1L<<20
digiOutDoesHW_DMA        =1L<<21
digiOutDoesHWPlayThru    =1L<<22
digiOutDoesILUT          =1L<<23
digiOutDoesKeyColor      =1L<<24
digiOutDoesAsyncGrabs     =1L<<25
digiOutDoesUnreadableScreenBits =1L<<26
digiOutDoesCompress      =1L<<27
digiOutDoesCompressOnly  =1L<<28
digiOutDoesPlayThruDuringCompress =1L<<29
digiOutDoesCompressPartiallyVisible =1L<<30
digiOutDoesNotNeedCopyOfCompressData =1L<<31

```

digiInDoesNTSC

The video digitizer supports National Television System Committee (NTSC) format input video signals. This flag is set to 1 if the digitizer component supports NTSC video.

digiInDoesPAL

The video digitizer component supports Phase Alternation Line (PAL) format input video signals. This flag is set to 1 if the digitizer component supports PAL video.

`digiInDoesSECAM`

The video digitizer component supports Systeme Electronique Couleur avec Memoire (SECAM) format input video signals. This flag is set to 1 if the digitizer component supports SECAM video.

`digiInDoesGenLock`

The video digitizer component supports genlock; that is, the digitizer can derive its timing from an external time base. This flag is set to 1 if the digitizer component supports genlock.

`digiInDoesComposite`

The video digitizer component supports composite input video. This flag is set to 1 if the digitizer component supports composite input.

`digitInDoesSVideo`

The video digitizer component supports s-video input video. This flag is set to 1 if the digitizer component supports s-video input.

`digiInDoesComponent`

The video digitizer component supports RGB input video. This flag is set to 1 if the digitizer component supports RGB input.

`digiInVTR_Broadcast`

The video digitizer component can distinguish between an input signal that emanates from a videotape player and a broadcast signal. This flag is set to 1 if the digitizer component can differentiate between the two different signal types.

`digiInDoesColor`

The video digitizer component supports color input. This flag is set to 1 if the digitizer component can accept color input.

`digiInDoesBW`

The video digitizer component supports grayscale input. This flag is set to 1 if the digitizer component can accept grayscale input.

`digiInSignalLock`

The video digitizer component is locked onto the input signal. If this flag is set to 1, the digitizer component detects either vertical or horizontal signal lock.

`digiOutDoes1`

The video digitizer component can work with pixel maps that contain 1-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 1-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

`digiOutDoes2`

The video digitizer component can work with pixel maps that contain 2-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 2-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

`digiOutDoes4`

The video digitizer component can work with pixel maps that contain 4-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 4-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

`digiOutDoes8`

The video digitizer component can work with pixel maps that contain 8-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 8-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

`digiOutDoes16`

The video digitizer component can work with pixel maps that contain 16-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 16-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

`digiOutDoes32`

The video digitizer component can work with pixel maps that contain 32-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 32-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

`digiOutDoesDither`

The video digitizer component supports **dithering**. If this flag is set to 1, the component supports dithering of colors. If this flag is set to 0, the digitizer component does not support dithering.

`digiOutDoesStretch`

The video digitizer component can stretch images to arbitrary sizes. If this flag is set to 1, the digitizer component can stretch images. If this flag is set to 0, the digitizer component does not support stretching.

`digiOutDoesShrink`

The video digitizer component can shrink images to arbitrary sizes. If this flag is set to 1, the digitizer component can shrink images. If this flag is set to 0, the digitizer component does not support shrinking.

`digiOutDoesMask`

The video digitizer component can handle clipping regions. If this flag is set to 1, the digitizer component can mask to an arbitrary clipping region. If this flag is set to 0, the digitizer component does not support clipping regions.

`digiOutDoesDouble`

The video digitizer component supports stretching to quadruple size when displaying the output video. The parameters for the stretch operation are specified in the matrix structure for the request; the component modifies the scaling attributes of the matrix (see the chapter “Movie Toolbox” in Inside Macintosh: QuickTime for information about transformation matrices). If this flag is set to 1, the digitizer component can stretch an image to exactly four times its original size, up to the maximum size specified by the `maxDestHeight` and `maxDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support stretching to quadruple size.

`digiOutDoesQuad`

The video digitizer component supports stretching an image to 16 times its original size when displaying the output video. The parameters for the stretch operation are specified in the matrix structure for the request; the component modifies the scaling attributes of the matrix (see the chapter “Movie Toolbox” in Inside Macintosh: QuickTime for information about transformation matrices). If this flag is set to 1, the digitizer component can stretch an image to exactly 16 times its original size, up to the maximum size specified by the `maxDestHeight` and `maxDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support this capability.

`digiOutDoesQuarter`

The video digitizer component can shrink an image to one-quarter of its original size when displaying the output video. The parameters for the shrink operation are specified in the matrix structure for the request; the component modifies the scaling attributes of the matrix (see the chapter “Movie Toolbox” in Inside Macintosh: QuickTime for information about transformation matrices). If this flag is set to 1, the digitizer component can shrink an image to exactly one-quarter of its original size, down to the minimum size specified by the `minDestHeight` and `minDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support this capability.

`digiOutDoesSixteenth`

The video digitizer component can shrink an image to 1/16 of its original size when displaying the output video. The parameters for the shrink operation are specified in the matrix structure for the request; the digitizer component modifies the scaling attributes of the matrix (see the chapter “Movie Toolbox”

in Inside Macintosh: QuickTime for information about transformation matrices). If this flag is set to 1, the digitizer component can shrink an image to exactly 1/16 of its original size, down to the minimum size specified by the `minDestHeight` and `minDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support this capability.

`digiOutDoesRotate`

The video digitizer component can rotate an image when displaying the output video. The parameters for the rotation are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can rotate the image. If this flag is set to 0, the digitizer component cannot rotate the resulting image.

`digiOutDoesHorizFlip`

The video digitizer component can flip an image horizontally when displaying the output video. The parameters for the horizontal flip are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can flip the image. If this flag is set to 0, the digitizer component cannot flip the resulting image.

`digiOutDoesVertFlip`

The video digitizer component can flip an image vertically when displaying the output video. The parameters for the vertical flip are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can flip the image. If this flag is set to 0, the digitizer component cannot flip the resulting image.

`digiOutDoesSkew`

The video digitizer component can skew an image when displaying the output video. Skewing an image distorts it linearly along only a single axis; for example, drawing a rectangular image into a parallelogram-shaped region. The parameters for the skew operation are specified in the matrix structure for the request. If this flag is set to 1, the digitizer component can skew an image. If this flag is set to 0, the digitizer component does not support this capability.

`digiOutDoesBlend`

The video digitizer component can blend the resulting image with a matte when displaying the output video. The matte is provided by the application by defining either an alpha channel or a mask plane. If this flag is set to 1, the digitizer component can blend. If this flag is set to 0, the digitizer component does not support this capability.

`digiOutDoesWarp`

The video digitizer component can warp an image when displaying the output video. Warping an image distorts it along one or more axes, perhaps nonlinearly, in effect “bending” the result region. The parameters for the warp

operation are specified in the matrix structure for the request. If this flag is set to 1, the digitizer component can warp an image. If this flag is set to 0, the digitizer component does not support this capability.

`digi0OutDoesDMA`

The video digitizer component can write to any screen or to offscreen memory. If this flag is set to 1, the digitizer component can use DMA to write to any screen or memory location.

`digi0OutDoesHWPlayThru`

The video digitizer component does not need idle time in order to display its video. If this flag is set to 1, your application does not need to grant processor time to the digitizer component at normal display speeds.

`digi0OutDoesILUT`

The video digitizer component supports inverse lookup tables for indexed color modes. If this flag is set to 1, the digitizer component uses inverse lookup tables when appropriate.

`digi0OutDoesKeyColor`

The video digitizer component supports clipping by means of key colors. If this flag is set to 1, the digitizer component can clip to a region defined by a key color.

`digi0OutDoesAsyncGrabs`

The video digitizer component can operate asynchronously. If this flag is set to 1, your application can use the `VDSetupBuffers` and `VDGrabOneFrameAsync` functions (described on page 0-669 and page 0-671, respectively).

`digi0OutDoesUnreadableScreenBits`

The video digitizer may place pixels on the screen that cannot be used when compressing images.

`digi0OutDoesCompress`

The video digitizer component supports compressed source devices. These devices provide compressed data directly, without having to use the Image Compression Manager. See “Controlling Compressed Source Devices” beginning on page 0-657 for more information about the functions that applications can use to work with compressed source devices.

`digi0OutDoesCompressOnly`

The video digitizer component only provides compressed image data; the component cannot provide displayable data. This flag only applies to digitizers that support compressed source devices.

`digiOutDoesPlayThruDuringCompress`

The video digitizer component can draw images on the screen at the same time that it is delivering compressed image data. This flag only applies to digitizers that support compressed source devices.

`digiOutDoesNotNeedCopyOfCompressData`

The sequence grabber does not make any copies of compressed data coming from a video digitizer during recording. This allows specific hardware to perform direct disk I/O without the need of a custom data handler.

Discussion

Video digitizer components report both their capabilities and their current status by returning a flags field that contains 1 bit for each of the capability flags. When reporting input status, for example, a video digitizer component sets the `digiInDoesGenLock` flag to 1 whenever the digitizer component is deriving its time signal from the input video. When reporting its input capabilities, the digitizer component sets this flag to 1 to indicate that it can derive its timing from the input video.

See Also

See `VDGetCurrentFlags` (III–1996), `VDGetDigitizerInfo` (III–1998).

Volume V

Programming Summary and Indexes

About Volume V

Volume V of *Inside QuickTime: API Reference* is designed to help you access and understand the information contained in Volumes I through IV. This volume contains the following sections:

- A programming summary, in which the main QuickTime functions and data structures are grouped and explained in terms of programming tasks.
- An index of data types that locates the documentation for each of QuickTime's structures, declared types, atom types, and callback pointers.
- An index of constants that locates each of QuickTime's enumerated constants, including atom codes.
- A list of functions sorted in the order in which their prototypes appear in the QuickTime header files.
- A bibliography that lists other books and websites containing information about QuickTime programming.
- A glossary of QuickTime-specific technical terms used in this reference.
- A history of changes to the QuickTime API and its documentation, starting with the first publication of this reference.

The *API Reference* as a whole is designed to let you absorb the same information from multiple viewpoints, using several media. To begin a coding task, for example, you might start with the programming summary in this volume. You could skim it in paper or PDF to get a feel for the kinds of functions that QuickTime provides for your task, and to see how they interrelate. Then you could switch to the online documentation of specific functions and data structures, clicking back and forth between detailed information and the programming summary. Any time you encountered an unfamiliar constant or data type, you could look it up in one of the indexes.

When using this five-volume work, however, remember that it is a *reference*; its primary purpose is to help you find information that you already know you need. For a more general understanding of QuickTime's capabilities and the structure of QuickTime code, consult the programming guides in the QuickTime Developer Series and QuickTime Technical Library. These books are listed in the bibliography that starts on page V–2911.

P R E F A C E T O V O L U M E V

Using Movie Toolbox Data Structures

Most QuickTime Movie Toolbox data structures are private. Your application never modifies the contents of these structures directly. Rather, the Movie Toolbox provides a number of functions that allow you to work with them.

Movie Identifiers

You identify various data structures to the Movie Toolbox by means of data types that are supplied by Movie Toolbox functions.

Data Types

Media

Specifies the media for an operation. Your application obtains a media identifier from such Movie Toolbox functions as `NewTrackMedia` (II–1120) and `GetTrackMedia` (I–535).

Movie

Specifies the movie for an operation. Your application obtains a movie identifier from such functions as `NewMovie` (II–1069), `NewMovieFromFile` (II–1080), and `NewMovieFromHandle` (II–1084).

MovieEditState

Specifies the movie edit state for an operation. Your application obtains a movie edit state identifier when you create the edit state by calling `NewMovieEditState` (II–1073).

QTCallback

Specifies the callback for an operation. You obtain a callback identifier from `NewCallback` (II–1053).

TimeBase

Specifies the time base for an operation. Your application obtains a time base identifier from `NewTimeBase` (II–1119) or `GetMovieTimeBase` (I–496).

Track

Specifies the track for an operation. Your application obtains a track identifier from such Movie Toolbox functions as `NewMovieTrack` (II–1092) and `GetMovieTrack` (I–497).

TrackEditState

Specifies the track edit state for an operation. Your application obtains a track edit state identifier when you create the edit state by calling `NewTrackEditState` (II–1119).

UserData

Specifies the **user data list** for an operation. You obtain a user data list identifier by calling `GetMovieUserData` (I–499), `GetTrackUserData` (I–546), or `GetMediaUserData` (I–456).

The Time Structure

The Movie Toolbox provides a number of functions that allow you to work with time specifications. Many of these functions require that you place a time specification in a data structure called a **time structure**. The time structure allows you to fully describe a time specification.

The data structure listed below defines the format of a **time structure**.

Data Structures

TimeRecord (IV–2486)

Contains a time value with its scale and time base.

The Fixed-Point and Fixed-Rectangle Structures

The Movie Toolbox matrix functions provide two mechanisms for specifying points and rectangles. Some of the functions work with standard QuickDraw points and rectangles, which use integer values to identify coordinates. Others, such as `TransformFixedRect` (III–1952), work with points and rectangles whose coordinates are expressed as fixed-point numbers. By using fixed-point numbers in these points and rectangles, the Movie Toolbox can support a greater degree of precision when defining graphic objects.

The `FixedPoint` (IV–2258) data type defines a fixed point. The `FixedRect` (IV–2260) data type defines a fixed rectangle. Note that both of these structures define the x coordinate before the y coordinate. In this they differ from standard QuickDraw structures.

Data Structures

`FixedPoint` (IV–2258)

Defines the position of a geometric point in fixed-point numbers.

`FixedRect` (IV–2260)

Defines the size and location of a rectangle in fixed-point numbers.

The Sound Description Structure

A sound description structure contains information that defines the characteristics of one or more sound samples. Data in the sound description structure indicates the type of compression that was used, the sample size, the rate at which samples were obtained, and so on. Sound media handlers use the information in the sound description structure when they process the sound samples.

Sound media handlers use the information in the sound description structure when they process the sound samples.

Data Structures

`SoundDescription` (IV–2449)

Describes the format of a collection of audio samples.

See Also

See “Sound Media Handler Functions” (V–2755) for more information about sound media handlers. See “Image Compression Data Structures” (V–2786) for a description of the image description structure, which contains information that defines the characteristics of an image.

Getting and Playing Movies

The Movie Toolbox provides a number of functions that allow applications to get and play movies. There are also a number of functions that allow you to create new movies.

Initializing the Movie Toolbox

The Movie Toolbox maintains state information for every application that is currently using the toolbox. The toolbox uses this information to keep track of the application's movies.

Before calling any other Movie Toolbox functions, your application must establish this working environment by calling `EnterMovies` (I-337). When your application is finished with the Movie Toolbox, you can release this storage by calling `ExitMovies` (I-340).

Functions

`EnterMovies` (I-337)

Initializes the Movie Toolbox and creates a private storage area for your application.

`ExitMovies` (I-340)

Automatically called when an application quits.

Error Functions

The Movie Toolbox provides a number of functions that allow your application to examine result codes generated by toolbox functions. In addition, the Movie Toolbox allows your application to provide a function that performs custom error notification.

In addition to returning errors as function results, the Movie Toolbox functions return error indications to calling applications by setting one of two values that are private to the Movie Toolbox: a current error value or a **sticky error** value.

The current error value contains the result code from the last Movie Toolbox function. The toolbox updates the current error value each time your application calls a Movie Toolbox function. Your application may call `GetMoviesError` (I-492) to obtain the current error value after calling any Movie Toolbox function. Many Movie Toolbox functions do not return an error as a function result; you must use `GetMoviesError` (I-492) to obtain the result code. Even if a function explicitly returns an error as a function result, that result is also available using `GetMoviesError` (I-492).

The Movie Toolbox saves a result code in the sticky error value. Your application clears the sticky error value by calling `ClearMoviesStickyError` (I-90). The Movie

Toolbox then places the first nonzero result code from any toolbox function used by your application into the sticky error value. The Movie Toolbox does not replace the value in the sticky error value until your application clears the value again. Your application uses `GetMoviesStickyError` (I-494) to obtain the result code stored in the sticky error value. In this manner, you can preserve and retrieve important result code information.

Your application can use `SetMoviesErrorProc` (III-1633) to designate an error function. The Movie Toolbox calls this error function each time there is an error.

Functions

`GetMoviesError` (I-492)

Returns the contents of the current error value and resets the current error value to 0.

`GetMoviesStickyError` (I-494)

Returns the contents of the **sticky error** value.

`ClearMoviesStickyError` (I-90)

Clears the **sticky error** value.

`SetMoviesErrorProc` (III-1633)

Performs custom error notification.

Callbacks

`MoviesErrorProc` (III-2109)

An error-notification function, called each time the current error value is to be set to a nonzero value.

Movie Functions

The Movie Toolbox provides a set of functions that allow your application to create, access, and convert movie files. Movie files contain data for QuickTime movies. You can also use the Movie Toolbox to load movies into memory, in preparation for working with the movie. These functions differ based on where the movie is stored.

Before your application can play a movie, you must first open the file that contains the movie. Your application can use `OpenMovieFile` (II-1133) to open a movie file. Once you are done with the file, your application releases the file by calling `CloseMovieFile` (I-102). Your application can create and open a new movie file by calling `CreateMovieFile` (I-145). Your application can delete a movie file by calling

DeleteMovieFile (I–249).

You can use NewMovie (II–1069) to create a new empty movie. If your application is loading a movie from an existing file, use either NewMovieFromFile (II–1080) or NewMovieFromDataFork (II–1075). NewMovieFromFile works with the file reference number you obtain from OpenMovieFile (II–1133). NewMovieFromDataFork (II–1075) works with movies stored in your document file’s data fork. Your application can then use the functions described in “Saving Movies” (V–2715) to load and store movies.

You can use ConvertFileToMovieFile (I–129) to specify an input file and convert it to a movie file. ConvertMovieToFile (I–134) takes a specified movie (or a single track within that movie) and converts it into an output file.

Once you are finished working with a movie, you should release the **resources** used by the movie by calling DisposeMovie (I–268).

Functions

NewMovieFromFile (II–1080)

Creates a new movie from certain file types that do not contain movie **resources** (AIFF, MPEG, etc).

NewMovieFromHandle (II–1084)

Creates a movie in memory from a movie **resource** or a handle you obtained from PutMovieIntoHandle (II–1151).

NewMovie (II–1069)

Creates a new movie in memory.

NewMovieFromUserProc (II–1087)

Creates a movie from data that you provide.

NewMovieFromDataRef (II–1078)

Creates a movie from any device with a corresponding data handler.

ConvertFileToMovieFile (I–129)

Converts a file to a movie file and supports a user settings dialog box for import operations.

ConvertMovieToFile (I–134)

Takes a specified movie (or a single track within that movie) and converts it into a specified file and type, supporting a Save As dialog box.

DisposeMovie (I-268)

Frees any memory being used by a movie, including the memory used by the movie's tracks and media structures.

CreateMovieFile (I-145)

Creates a movie file, creates an empty movie which references the file, and opens the movie file with write permission.

OpenMovieFile (II-1133)

Opens a specified movie file.

CloseMovieFile (I-102)

Closes an open movie file.

DeleteMovieFile (I-249)

Deletes a movie file.

Saving Movies

The Movie Toolbox provides a set of high-level functions for storing movies within files. These files have a file type of 'MooV' and a **resource** type of 'moov'. Your application can gain access to existing movies with either `NewMovieFromFile` (II-1080) or `NewMovieFromDataFork` (II-1075). Once you have loaded the movie, your application uses the functions described below to save any changes you have made to the movie.

You can use `AddMovieResource` (I-36) to add a new movie **resource** to a movie file. Your application can use this function to save a movie that it created using the functions described in “Editing Movies” (V-2746). You can use the `UpdateMovieResource` (III-1983) function to replace an existing movie resource in a movie file. You can remove a movie resource by calling the `RemoveMovieResource` (III-1458) function.

The movie resources that your application creates with `AddMovieResource` (I-36) and `UpdateMovieResource` (III-1983) may contain references to movie data. These references identify the data that constitute the movie. However, the movie data can be stored outside of the movie file. If you want to create a movie file that contains all of its movie data, use `FlattenMovie` (I-372) or `FlattenMovieData` (I-374). These functions can also be used to store the movie data in the movie file's data fork, or to interleave the media data to optimize performance.

`PutMovieIntoHandle` (II-1151) places a QuickTime movie into a handle. You can then

convert the movie into specialized data formats.

`HasMovieChanged` (I–666) and `ClearMovieChanged` (I–89) allow your application to work with the movie changed flag that is maintained by the Movie Toolbox. You can use this flag to determine whether a movie has been changed.

The movie changed flag indicates whether you have changed the movie. Such actions as editing the movie, adding samples to a media, or changing a data reference cause the flag to indicate that the movie has changed. There are several operations that the movie changed flag does not reflect, including changing the volume, rate, or time settings for the movie. These settings change frequently when a movie is played. Your application must monitor these settings itself.

The Movie Toolbox also supplies functions for storing and retrieving movies that are stored in the data fork of a file. These functions provide robust data reference resolution and improve low memory performance. `NewMovieFromDataFork` (II–1075) enables you to retrieve a movie that is stored anywhere in the data fork of a file. You can use `PutMovieIntoDataFork` (II–1150) to store an atom version of a specified movie in the data fork of a file.

Functions

`HasMovieChanged` (I–666)

Determines whether a movie has changed and needs to be saved.

`ClearMovieChanged` (I–89)

Sets the movie changed flag to indicate that the movie has not been changed.

`AddMovieResource` (I–36)

Adds a movie **resource** to a specified resource file.

`UpdateMovieResource` (III–1983)

Replaces the contents of a movie **resource** in a specified movie file.

`RemoveMovieResource` (III–1458)

Removes a movie **resource** from a specified movie file.

`PutMovieIntoHandle` (II–1151)

Creates a new movie **resource**.

`FlattenMovie` (I–372)

Creates a new movie file containing a specified movie.

`FlattenMovieData` (I–374)

Creates a new movie and a file that contains all the movie data.

`NewMovieFromDataFork` (II–1075)

Retrieves a movie that is stored anywhere in the data fork of a specified Macintosh file.

`PutMovieIntoDataFork` (II–1150)

Stores a movie in the data fork of a given file.

Controlling Movie Playback

A number of high-level functions provided by the Movie Toolbox allow your application to play movies.

You can use the `StartMovie` (III–1911) and `StopMovie` (III–1914) functions to start and stop movies. You can use `GoToBeginningOfMovie` (I–550) and `GoToEndOfMovie` to set the position at either the beginning or the end of a movie.

Functions

`StartMovie` (III–1911)

Starts the movie playing from the current movie time.

`StopMovie` (III–1914)

Stops the playback of a movie.

`GoToBeginningOfMovie` (I–550)

Repositions a movie to play from its start.

`GoToEndOfMovie` (I–551)

Repositions a movie to play from its end.

See Also

For information about how to control a movie’s playback rate, see “Working With Movie Time” (V–2732). Functions that work with time bases, such as `SetMovieTimeValue` (III–1635) and `GetMovieTimeScale` (I–496), can be used to control the current position anywhere within a movie.

Using the Full Screen

Two functions let you put a device into full-screen mode (that is, select where and when the menu bar is not visible).

These functions are listed below.

Functions

`BeginFullScreen` (I–52)

Begins full-screen mode for a specified monitor.

`EndFullScreen` (I–314)

Ends full-screen mode for a graphics device.

Controlling Hardware Scaling

Three functions that allow applications to zoom a monitor. These functions are considered low-level calls that you should use only when playing back QuickTime movies in a controlled environment with no user interaction. Also, because this capability is not present on all computers, applications should not depend on its availability.

These functions are listed below.

Functions

`GDHasScale` (I–381)

Returns the closest possible scaling that a particular screen device can be set to in a given pixel depth.

`GDGetScale` (I–380)

Returns the current scale of the given screen graphics device.

`GDSetscale` (I–382)

Sets a screen graphics device to a new scale.

Movie Posters and Movie Previews

A QuickTime movie may contain a **preview** and a **poster**. A movie preview is a very short version of a movie, typically less than five seconds in duration. The preview is intended to give the user an idea of a movie's contents. A movie poster is a still frame representing the movie.

Use `PlayMoviePreview` (II–1140) to display a movie's **preview**. `PlayMoviePreview` (II–1140) sets the movie into preview mode, plays the movie preview, sets the movie back to normal playback mode, and returns to your application.

Alternatively, your application can control the playback of a movie's preview. Use

SetMoviePreviewMode (III–1628) to place a movie into preview mode. You can then use StartMovie (III–1911) and StopMovie (III–1914) to control movie playback. Your application can find out if a movie is in preview mode by calling GetMoviePreviewMode (I–487).

Your application can specify the starting time and duration of the movie preview with SetMoviePreviewTime (III–1629) and GetMoviePreviewTime.

Use ShowMoviePoster (III–1827) to display a movie's **poster**. You can work with the poster's boundary rectangle using SetPosterBox (III–1638) and GetPosterBox (I–505). Your application can work with the starting time of the poster with SetMoviePosterTime (III–1625) and GetMoviePosterTime. Posters always have no duration.

Tracks may be specified for use in the movie, its preview, its poster, or any combination of the three. So, for example, when the Movie Toolbox plays the movie preview it uses only those tracks that are assigned to the preview. Your application controls the use of a movie's tracks with SetTrackUsage (III–1668). You can find out how a track is used by calling GetTrackUsage (I–545).

Functions

SetTrackUsage (III–1668)

Specifies whether a track is used in a movie, its **preview**, its **poster**, or a combination of these.

GetTrackUsage (I–545)

Determines whether a track is used in a movie, its **preview**, its **poster**, or a combination of these.

ShowMoviePoster (III–1827)

Displays a movie's **poster**.

SetPosterBox (III–1638)

Sets a **poster**'s boundary rectangle.

GetPosterBox (I–505)

Obtains a **poster**'s boundary rectangle.

SetMoviePosterTime (III–1625)

Sets the **poster** time for the movie.

GetMoviePosterTime (I–485)

Returns the **poster**'s time in a movie.

PlayMoviePreview (II–1140)

Plays a movie's **preview**.

SetMoviePreviewMode (III–1628)

Places a movie into and out of **preview** mode.

GetMoviePreviewMode (I–487)

Determines whether a movie is in **preview** mode.

SetMoviePreviewTime (III–1629)

Defines the starting time and duration of the movie's **preview**.

GetMoviePreviewTime (I–487)

Returns the starting time and duration of the movie's **preview**.

Movies and Your Event Loop

The Movie Toolbox supplies several functions that your application normally calls from its main event loop.

In order for your movies to play, your application must grant time to the Movie Toolbox. You do this by calling `MoviesTask` (II–973) from your main event loop. `MoviesTask` (II–973) causes the Movie Toolbox to service all your active movies. You should call this function regularly so that your movie can play smoothly. You can use `UpdateMovie` (III–1981) to force your movie to be redrawn after it has been uncovered.

You may want your application to take a particular action when a movie is done playing. The Movie Toolbox provides `IsMovieDone` (II–774), which allows you to determine whether a movie is done playing.

The Movie Toolbox provides two functions that allow your application to determine whether a specified point lies in either a movie or a track. Use `PtInMovie` (II–1148) with movies; use `PtInTrack` (II–1149) with tracks.

Your application can retrieve some status information about movies and tracks. Use `GetMovieStatus` (I–494) to retrieve movie status; use `GetTrackStatus` (I–544) to get track status.

Functions

`MoviesTask` (II–973)

Serves active movies.

IsMovieDone (II–774)

Determines if a particular movie has completely finished playing.

UpdateMovie (III–1981)

Ensures that the Movie Toolbox properly displays your movie after it has been uncovered.

InvalidateMovieRegion (II–771)

Invalidates a small area of a movie.

PtInMovie (II–1148)

Determines whether a specified point lies in the region defined by a movie's final display boundary region after it has been clipped by the movie's display clipping region.

PtInTrack (II–1149)

Determines whether a specified point lies in the region defined by a track's display boundary region after it has been clipped by the movie's final display clipping region.

GetMovieStatus (I–494)

Searches for errors in all the enabled tracks of the movie and returns information about errors that are encountered during the processing associated with the `MoviesTask` function.

GetTrackStatus (I–544)

Returns the value of the last error the media encountered while playing a specified track.

See Also

The Movie Toolbox also provides more sophisticated callback mechanisms, which are discussed in “Time Base Callback Functions” (V–2763).

Preferred Movie Settings

Every movie has default, or preferred, settings for playback rate and volume. These settings are stored with the movie in its movie file. The Movie Toolbox provides functions that allow your application to manipulate these default settings.

You can use `GetMoviePreferredRate` (I–485) and `SetMoviePreferredRate` to work with a movie's default playback rate. You can use `GetMoviePreferredVolume` (I–486) and `SetMoviePreferredVolume` to work with the default sound volume of a movie.

Functions

`GetMoviePreferredRate` (I–485)

Returns a movie's default playback rate.

`SetMoviePreferredRate` (III–1626)

Specifies a movie's default playback rate.

`GetMoviePreferredVolume` (I–486)

Returns a movie's preferred volume setting.

`SetMoviePreferredVolume` (III–1627)

Sets a movie's preferred volume setting.

See Also

You can use `SetMovieRate` to change a movie's playback rate. The Movie Toolbox also provides a number of functions that allow you to change other settings when you play a movie. They are discussed in “Modifying Movie Properties” (V–2728).

Enhancing Movie Playback Performance

There are circumstances in which an application needs to optimize the performance of a movie or a portion of a movie. The Movie Toolbox provides several functions to help in this process.

The first step you can take to enhance movie playback performance is to allow the Movie Toolbox to preroll the movie. When the toolbox prerolls a movie, it informs the media handlers that the movie is about to play. The media handlers can then load the appropriate movie data. In this manner, the movie can play smoothly from the start. Use `PrerollMovie` (II–1142) to preroll a movie.

The next performance enhancement technique is to load portions of a movie, track, or media into memory, thus reducing or eliminating disk access during playback. Loading the movie into RAM provides most noticeable performance improvements when there is a lot of random access involved in the playback process and the entire movie fits into available memory. Use `LoadMovieIntoRam` (II–781), `LoadTrackIntoRam`, and `LoadMediaIntoRam` (II–779) to copy all or part of a movie into memory. These functions load tracks into memory in a time-slice order so that, if a function fails because it is out of memory, all tracks are left loaded to about the same point in time.

You can influence the temporal accuracy, and therefore the speed, with which the Movie Toolbox tries to display a movie by calling either `SetMoviePlayHints` (III–1623) or `SetMediaPlayHints`.

For each movie currently in use, the Movie Toolbox maintains an active movie segment. The active movie segment is the part of the movie that your application is interested in playing. By default, the active movie segment is set to be the entire movie. You may wish to change this to be some segment of the movie; for example, if you wish to play a user's selection repeatedly. By setting the active movie segment you guarantee that the Movie Toolbox uses no samples from outside of that range while playing the movie. Use `GetMovieActiveSegment` (I-457) and `SetMovieActiveSegment` to work with the active segment.

Some movies contain very few **key frames** and a great number of frame differences. These movies play back very well because they have a lower data rate. Unfortunately, this makes random access operations on a movie, such as scrubbing, difficult. To improve random access performance of movies with few key frames and many frame differences, shadow **sync samples** may be added. Shadow sync samples are self-contained samples that are alternates for already existing frame difference samples. During certain random access operations, a **shadow sync** sample is used instead of a normal key frame, which may be very far away from the desired frame. The Movie Toolbox provides two functions to let you create just such an association between a frame difference sample and a sync sample. `SetMediaShadowSync` (III-1607) establishes a shadow sync sample for a media. You can use `GetMediaShadowSync` (I-453) to find out if a particular frame difference sample has a shadow sync sample.

Functions

`PrerollMovie` (II-1142)

Prepares a portion of a movie for playback.

`PrePrerollMovie` (II-1141)

Sets up any necessary network connections to receive streaming content.

`AbortPrePrerollMovie` (I-21)

Terminates the operation of `PrePrerollMovie` (II-1141).

`LoadMovieIntoRam` (II-781)

Loads a movie's data into memory.

`LoadTrackIntoRam` (II-782)

Loads a track's data into memory.

`LoadMediaIntoRam` (II-779)

Loads a media's data into memory.

SetMoviePlayHints (III–1623)

Provides information to the Movie Toolbox that can influence movie playback.

SetMediaPlayHints (III–1601)

Provides information to the Movie Toolbox that can influence playback of a single media.

GetMovieActiveSegment (I–457)

Determines what portion of a movie is currently active for playing.

SetMovieActiveSegment (III–1609)

Defines a movie’s active segment.

SetMediaShadowSync (III–1607)

Creates an association between the indicated frame difference sample and a specified self-contained sample in a given media.

GetMediaShadowSync (I–453)

Returns the sample number of the shadow **sync sample** associated with a given frame difference sample number.

SetTrackLoadSettings (III–1661)

Specifies a portion of a track that is to be loaded into memory whenever it is played.

GetTrackLoadSettings (I–532)

Retrieves a track’s preload information.

GetTrackDisplayMatrix (I–528)

Returns a matrix that is the concatenation of all matrices currently affecting the track’s location, scaling, and so on, including the movie’s matrix, the track’s matrix, and the modifier matrix.

Disabling Movies and Tracks

The Movie Toolbox services only movies and tracks that are active. It provides functions that allow your application to enable and disable tracks and movies.

You can use `SetMovieActive` (III–1609) to activate and deactivate a movie and `GetMovieActive` (I–456) to determine whether a movie is active. Similarly, your application can use `SetTrackEnabled` (III–1658) to enable and disable a track and `GetTrackEnabled` (I–531) to determine whether a track is enabled.

Functions

SetMovieActive (III–1609)

Activates or deactivates a movie.

GetMovieActive (I–456)

Determines whether a movie is currently active.

SetTrackEnabled (III–1658)

Enables or disables a track.

GetTrackEnabled (I–531)

Determines whether a track is currently enabled.

See Also

The Movie Toolbox also allows you to assign alternate tracks based on language or quality criteria. Functions that work with alternate tracks are discussed in “Working With Alternate Tracks” (V–2738).

Generating Pictures From Movies

The Movie Toolbox provides a set of functions that allow your application to create QuickDraw pictures from movies, tracks, and **posters**.

You can use GetMoviePict (I–483) to create a picture from a movie or its **preview**; you can use GetTrackPict (I–540) to create a picture from a track.

GetMoviePosterPict (I–484) lets you create a picture that contains a movie’s **poster**. If a movie or track has no spatial representation, the returned picture is empty (that is, the upper-left and lower-right coordinates are equal).

Functions

GetMoviePict (I–483)

Creates a QuickDraw picture from a specified movie at a specified time.

GetTrackPict (I–540)

Creates a QuickDraw picture from a specified track at a specified time.

GetMoviePosterPict (I–484)

Creates a QuickDraw picture that contains a movie’s **poster**.

Creating Tracks and Media Structures

The Movie Toolbox provides several functions that allow your application to create new movie tracks and media structures and to dispose of existing tracks and media structures. You use these functions when you are creating a new movie or when you are editing an existing movie.

You can use `NewMovieTrack` (II–1092) to create a new track for a specified movie. Conversely, you can use `DisposeMovieTrack` (I–278) to dispose of an existing track. Your application can create a new media for a track by calling `NewTrackMedia` (II–1120). You can use `DisposeTrackMedia` (I–301) to dispose of an existing media.

Functions

`NewMovieTrack` (II–1092)

Creates a new movie track, without a media.

`DisposeMovieTrack` (I–278)

Removes a track from a movie.

`NewTrackMedia` (II–1120)

Creates a media for a new track.

`DisposeTrackMedia` (I–301)

Removes a media from a track.

Working With Progress and Cover Functions

The Movie Toolbox allows your application to assign two types of custom functions: **progress functions** and cover functions. These functions allow you to perform special processing under certain circumstances.

Some Movie Toolbox functions can take a long time to execute. For example, if you call `FlattenMovie` (I–372) and specify a large movie, the Movie Toolbox must read and write all the sample data for the movie. During such operations you may wish to display some kind of progress indicator to the user. A **progress function** is an application-defined function that you can use to track the progress of time-consuming activities, and thereby keep the user informed about that progress. The Movie Toolbox calls your progress function at regular intervals during long operations. The Movie Toolbox determines whether to call your function based on the duration of the operation; your function will not be called unnecessarily. When it calls your function, the Movie Toolbox provides information about the operation

that is underway and its relative completion. You can use this information to display an informational dialog box to the user.

The Movie Toolbox allows your application to perform custom processing whenever one of your movie's tracks covers a screen region or reveals a region that was previously covered. You perform this processing in cover functions. There are two types of cover functions: those that are called when your movie covers a screen region, and those that are called when your movie uncovers a screen region that was previously covered. Cover functions that are called when your movie covers a screen region are responsible for erasing the region; you may choose to save the hidden region in an offscreen buffer. Cover functions that are called when your movie reveals a hidden screen region must redisplay the hidden region. The Movie Toolbox sets the graphics world before it calls your cover function; your function must not change the graphics world. Note that the Movie Toolbox does not call your cover function in response to changes to the movie's transformation matrix.

The `SetMovieProgressProc` (III-1630) function helps your application work with progress functions and the `SetMovieCoverProcs` (III-1614) function helps your application work with cover functions. You should assign your progress function when you open the movie; the Movie Toolbox will call your function when it is appropriate to do so. One progress function may support more than one movie. When the Movie Toolbox calls your function, it provides you with the movie identifier so that you can discriminate between various movies.

Functions

`SetMovieProgressProc` (III-1630)

Attaches a **progress function** to a movie.

`SetMovieCoverProcs` (III-1614)

Sets the callbacks invoked when a movie is covered or uncovered.

`GetMovieCoverProcs` (I-464)

Retrieves the cover functions that you set with the `SetMovieCoverProcs` function.

`SetMovieDrawingCompleteProc` (III-1617)

Assigns a drawing-complete function to a movie.

Callbacks

`MovieProgressProc` (III-2105)

Monitors the progress of the Movie Toolbox during long operations.

MovieDrawingCompleteProc (III–2100)

Called when movie drawing is complete.

See Also

The Movie Toolbox provides default cover functions. When your movie uncovers a region, the default function that is called erases the movie’s image by displaying the graphics port’s background color and pattern. You can set the port’s characteristics by calling `SetMovieGWorld`. When your movie covers a region, the default function that is called does nothing.

Modifying Movie Properties

The Movie Toolbox provides a number of functions that allow applications to edit existing movies or to create the contents of new movies.

Working With Movie Spatial Characteristics

The Movie Toolbox provides a number of functions that allow your application to determine and change the display characteristics of movies and tracks.

Your application can work with a movie’s matrix by calling `GetMovieMatrix` (I–478) and `SetMovieMatrix` (III–1622), and it can work with a track’s matrix with `GetTrackMatrix` (I–533) `SetTrackMatrix`. Then you can perform operations on matrixes with the Movie Toolbox’s matrix functions described in “Working With Matrices” (V–2764).

Several functions affect the displayed movie and its tracks in the final display coordinate system. `SetMovieGWorld` (III–1619) and `GetMovieGWorld` (I–471) let you work with a movie’s display destination. `GetMovieBox` (I–460) and `SetMovieBox` (III–1611) allow you to work with a movie’s boundary rectangle and its associated transformations. Alternatively, you can work directly with a movie’s transformation matrix. `GetMovieDisplayBoundsRgn` (I–468) determines a movie’s boundary region at the current movie time. On the other hand, `GetMovieSegmentDisplayBoundsRgn` (I–490) determines a movie’s boundary region over a specified time segment. You can use `GetMovieDisplayClipRgn` (I–469) and `SetMovieDisplayClipRgn` to work with a movie’s display clipping region.

`GetTrackDisplayBoundsRgn` (I–528) and `GetTrackSegmentDisplayBoundsRgn` determine a track's final boundary region. You can use `GetTrackLayer` (I–532) and `SetTrackLayer` (III–1660) to control the drawing order of tracks within a movie.

A number of functions affect a movie's display boundaries before any display transformations; these functions operate in the movie's display coordinate system. You can use `GetMovieClipRgn` (I–462) and `SetMovieClipRgn` (III–1612) to work with a movie's clipping region (that is, the clipping region that is applied before the movie display transformation). Use `GetMovieBoundsRgn` (I–459) to determine a movie's boundary region at the current movie time.

Use `GetTrackMovieBoundsRgn` (I–537) to work with a track's boundary region after matrix transformations have placed the track into the movie's display system.

The Movie Toolbox provides several functions that affect a track's display boundaries; these functions operate in the track's display coordinate system before any other display transformations are applied. `GetTrackDimensions` (I–527) and `SetTrackDimensions` allow you to establish a track's coordinate system and to establish a track's source rectangle. A track's source rectangle defines the coordinate system of the track. You specify the dimensions of the rectangle by providing the coordinates of the lower-right corner of the rectangle. The Movie Toolbox sets the upper-left corner to (0,0) in the track's coordinate system.

You can use `GetTrackBoundsRgn` (I–523) to determine a track's boundary region. `GetTrackClipRgn` (I–524) and `SetTrackClipRgn` (III–1656) let you work with a track's clipping region.

You can use `GetTrackMatte` (I–534) and `SetTrackMatte` (III–1663) to establish a track's matte. `DisposeMatte` (I–267) function allows you to dispose of a matte once you are finished with it.

Functions

`GetMovieMatrix` (I–478)

Retrieves a movie's transformation matrix.

`SetMovieMatrix` (III–1622)

Sets a movie's transformation matrix.

`GetTrackMatrix` (I–533)

Retrieves a track's transformation matrix.

`SetTrackMatrix` (III–1662)

Establishes a track's transformation matrix.

`SetMovieGWorld` (III–1619)

Establishes a movie's display coordinate system by setting the graphics world for displaying the movie.

`GetMovieGWorld` (I–471)

Returns a movie's graphics world.

`GetMovieBox` (I–460)

Returns a movie's boundary rectangle, which is a rectangle that encompasses all of the movie's enabled tracks.

`SetMovieBox` (III–1611)

Sets a movie's boundary rectangle.

`GetMovieDisplayBoundsRgn` (I–468)

Determines a movie's display boundary region.

`GetMovieSegmentDisplayBoundsRgn` (I–490)

Determines a movie's display boundary region for a specified segment.

`GetMovieDisplayClipRgn` (I–469)

Determines a movie's current display clipping region.

`SetMovieDisplayClipRgn` (III–1616)

Establishes a movie's current display clipping region.

`SetMovieColorTable` (III–1613)

Associates a `ColorTable` (IV–2210) structure with a movie.

`GetMovieColorTable` (I–463)

Retrieves a movie's color table.

`GetTrackDisplayBoundsRgn` (I–528)

Determines the region a track occupies in a movie's graphics world.

`GetTrackSegmentDisplayBoundsRgn` (I–542)

Determines the region a track occupies in a movie's graphics world during a specified segment.

`GetTrackLayer` (I–532)

Retrieves a track's layer.

`SetTrackLayer` (III–1660)

Sets a track's layer.

GetMovieClipRgn (I-462)

Determines a movie's clipping region.

SetMovieClipRgn (III-1612)

Establishes a movie's clipping region.

GetMovieBoundsRgn (I-459)

Determines a movie's boundary region.

GetTrackMovieBoundsRgn (I-537)

Determines the region the track occupies in a movie's boundary region.

GetTrackDimensions (I-527)

Determines a track's source rectangle.

SetTrackDimensions (III-1657)

Establishes a track's source rectangle.

GetTrackBoundsRgn (I-523)

Lets the media limit the size of a track boundary rectangle.

GetTrackClipRgn (I-524)

Determines the clipping region of a track.

SetTrackClipRgn (III-1656)

Sets the clipping region of a track.

GetTrackMatte (I-534)

Retrieves a copy of a track's matte.

SetTrackMatte (III-1663)

Sets a track's matte.

DisposeMatte (I-267)

Disposes of a matte obtained from the GetTrackMatte (I-534) function.

SetTrackGWorld (III-1659)

Forces a track to draw into a particular graphics world, which may be different from that of the movie.

See Also

Before using any of these functions, you should be familiar with the way in which the Movie Toolbox displays movies. For this information, see *Inside QuickTime: Interactive Movies* (listed in the **bibliography**). Also see *Inside Macintosh: Imaging With QuickDraw* (listed in the **bibliography**) for detailed information about graphics worlds.

Working With Sound Volume

The Movie Toolbox allows you to set the sound volume of movies and tracks. Track volumes allow tracks within a movie to have different volumes. A track's volume is scaled by the movie's volume to produce the track's final volume. Finally, the movie's volume is scaled by the user's sound volume controls.

Volume values range from -1.0 to +1.0. Higher values translate to louder volume. Negative values indicate muted volume. That is, the Movie Toolbox does not play any sound for movies or tracks with negative volume settings, but the original volume level is retained as the absolute value of the volume setting. Therefore, if you want to toggle the current state of the volume, you can invert the sign of the current volume setting.

You can use `GetMovieVolume` (I-500) and `SetMovieVolume` (III-1637) to work with a movie's volume. `GetTrackVolume` (I-547) and `SetTrackVolume` (III-1669) allow you to work with a track's volume.

Functions

`GetMovieVolume` (I-500)

Returns a movie's current volume setting.

`SetMovieVolume` (III-1637)

Sets a movie's current volume.

`GetTrackVolume` (I-547)

Returns a track's current volume setting.

`SetTrackVolume` (III-1669)

Sets a track's current volume.

Working With Movie Time

Every QuickTime movie has its own time base. A movie's time base allows all the tracks that make up the movie to be synchronized when the movie is played. The Movie Toolbox provides a number of functions that allow your application to determine and establish the time parameters of a movie.

You can use `GetMovieTimeBase` (I-496) to retrieve the time base for a movie. You can work with a movie's current time by calling `GetMovieTime` (I-495), `SetMovieTime` (III-1634), and `SetMovieTimeValue` (III-1635). You can work with a movie's time

scale by calling `GetMovieTimeScale` (I–496) and `SetMovieTimeScale` (III–1635).

The Movie Toolbox can calculate the total duration of a movie; use `GetMovieDuration` (I–470) to retrieve a movie’s duration.

Your application can call `GetMovieRate` (I–490) and `SetMovieRate` (III–1631) to work with a movie’s playback rate.

Functions

`GetMovieTimeBase` (I–496)

Returns a movie’s time base.

`GetMovieTime` (I–495)

Returns a movie’s current time both as a time value and in a **time structure**.

`SetMovieTime` (III–1634)

Changes a movie’s current time.

`SetMovieTimeValue` (III–1635)

Sets a movie’s time value.

`GetMovieTimeScale` (I–496)

Returns the time scale of a movie.

`SetMovieTimeScale` (III–1635)

Establishes a movie’s time scale.

`GetMovieDuration` (I–470)

Returns the duration of a movie.

`GetMovieRate` (I–490)

Returns a movie’s playback rate.

`SetMovieRate` (III–1631)

Sets a movie’s playback rate.

See Also

See also “Working With Track Time” (V–2733) and “Working With Media Time” (V–2735).

Working With Track Time

The Movie Toolbox provides several functions that allow your application to determine and establish a track’s time parameters. A track uses the time base of the

movie that contains the track; therefore there are no functions that work with a track's time base or time scale. However, you can determine a track's duration and its offset from the start of a movie.

All of the tracks in a movie use the movie's time coordinate system. That is, the movie's time scale defines the basic time unit for each of the movie's tracks. Each track begins at the beginning of the movie, but the track's data might not begin until some time value other than 0. This intervening time is represented by blank space. In an audio track the blank space translates to silence; in a video track the blank space generates no visual image. This blank space is the track offset. Each track has its own duration. This duration need not correspond to the duration of the movie. A movie duration always equals the maximum track duration.

You can use `GetTrackDuration` (I-529) to determine a track's duration. `SetTrackOffset` (III-1664) and `GetTrackOffset` (I-539) enable you to work with a track's offset from the start of the movie that contains it. `TrackTimeToMediaTime` (III-1951) lets you translate a track's time to the corresponding time value of a media in the track.

Functions

`GetTrackDuration` (I-529)

Returns the duration of a track.

`SetTrackOffset` (III-1664)

Modifies the duration of the empty space that lies at the beginning of a track, thus changing the duration of the entire track.

`GetTrackOffset` (I-539)

Determines the time difference between the start of a track and the start of the movie that contains the track.

`TrackTimeToMediaTime` (III-1951)

Converts a track's time value to a time value that is appropriate to the track's media, using the track's edit list.

See Also

See also "Working With Movie Time" (V-2732) and "Working With Media Time" (V-2735).

Working With Media Time

The Movie Toolbox provides functions that allow your application to work with the time parameters of a media.

You can use `GetMediaDuration` (I–430) to determine a media’s duration. `GetMediaTimeScale` (I–455) and `SetMediaTimeScale` (III–1608) let you determine or establish a media’s time scale.

Functions

`GetMediaDuration` (I–430)

Returns the duration of a media.

`GetMediaTimeScale` (I–455)

Determines a media’s time scale.

`SetMediaTimeScale` (III–1608)

Sets a media’s time scale.

See Also

See also “Working With Track Time” (V–2733) and “Working With Movie Time” (V–2732).

Finding Interesting Times

The Movie Toolbox provides a set of functions that help you locate samples in movies, tracks, and media structures. These functions are based on the concept of interesting times. An interesting time refers to a time value in a movie, track, or media that meets certain search criteria. You specify the search criteria to the Movie Toolbox. The Movie Toolbox then scans the movie, track, or media, and locates time values that meet those search criteria. You can use these functions to search through image sequences. For example, you may want to locate each frame in an image sequence. Or you may be more interested in **key frames**, especially if you are trying to optimize display performance. An easy way to determine whether a movie has been edited is to look for track edits in the movie data. You may also be interested in searching for samples in a movie’s media. If you set the appropriate search criteria, the Movie Toolbox locates the appropriate frames for you. You need the functions described in this section because QuickTime doesn’t have a fixed rate. Each frame can have its own duration.

The Movie Toolbox identifies an interesting time by specifying its starting time and duration. The starting time indicates the time in the movie, track, or media where

the search criteria are met. The duration indicates the length of time during which the search criteria remain in effect. For example, if you are looking for samples in a media, the start time would indicate the beginning of the sample, and the duration would indicate the length of time to the next sample. In this case, you could find the next media sample by adding the duration to the start time. These duration values are always positive; you determine the direction of the search by setting the sign of the rate value you supply to the functions.

Note that movie interesting times are defined in the scope of the movie as a whole. As a result, one interesting time ends when another interesting time starts in any track in the movie. For example, if you are looking for **key frames** in a movie, the duration value from one interesting time tells you when the next key frame starts. However, that second key frame may be in a different track in the movie. Therefore, the duration of the interesting time does not necessarily correspond to the duration of the key frame.

You can use `GetMovieNextInterestingTime` (I-480) to locate times of interest in a movie. `GetTrackNextInterestingTime` (I-537) lets you work with tracks. Use `GetMediaNextInterestingTime` (I-437) to locate samples in a media.

Functions

`GetMovieNextInterestingTime` (I-480)

Searches for times of interest in a movie's enabled tracks.

`GetTrackNextInterestingTime` (I-537)

Searches for times of interest in a track.

`GetMediaNextInterestingTime` (I-437)

Searches for times of interest in a media.

Locating a Movie's Tracks and Media Structures

The Movie Toolbox provides a set of functions that help your application locate a movie's tracks and media structures. The Movie Toolbox identifies a movie's tracks in two ways. First, every track in a movie has a unique ID value. This ID value is unique throughout the life of a movie, even after it has been saved. That is, no two tracks of a movie ever have the same ID, and no ID value is ever reused. Second, a movie's current tracks may be identified by their index value. Index values always range from 1 to the number of tracks in the movie. Track indexes provide a convenient way to access each track of a movie.

There are several functions that allow you to find a movie's tracks. You can use `GetMovieTrackCount` (I-498) to determine the number of tracks in a movie. Use `GetMovieTrack` (I-497) to obtain the track identifier for a specific track, given its ID. `GetMovieIndTrack` (I-473) lets you obtain a track's identifier, given its track index.

You can obtain a track's ID value given its track identifier by calling `GetTrackID` (I-531). You can determine the movie that contains a track by calling `GetTrackMovie` (I-536).

`GetTrackMedia` (I-535) enables you to find a track's media. Conversely, you can find the track that uses a media by calling `GetMediaTrack` (I-455).

Functions

`GetMovieTrackCount` (I-498)

Returns the number of tracks in a movie.

`GetMovieTrack` (I-497)

Determines the track identifier of a track, given the track's ID value.

`GetMovieIndTrack` (I-473)

Determines the track identifier of a track, given the track's index value.

`GetMovieIndTrackType` (I-475)

Searches for all of a movie's tracks that share a given media type or media characteristic.

`GetTrackID` (I-531)

Determines a track's unique track ID value.

`GetTrackMovie` (I-536)

Determines the movie that contains a specified track.

`GetTrackMedia` (I-535)

Determines the media that contains a track's sample data.

`GetMediaTrack` (I-455)

Determines the track that uses a specified media.

Working With Track References

Track references allow you to relate tracks to one another. For example, they can help you identify the text track that contains the subtitles for a movie's audio track and relate that text track to a particular audio track.

The `AddTrackReference` (I–42) function allows you to relate one track to another. The `DeleteTrackReference` (I–252) function removes that relationship. The `SetTrackReference` (III–1665) and `GetTrackReference` (I–541) functions allow you to modify an existing track reference so that it identifies a different track. The `GetNextTrackReferenceType` (I–503) and `GetTrackReferenceCount` functions allow you to scan all of a track’s track references.

Functions

`AddTrackReference` (I–42)

Adds a new track reference to a track.

`DeleteTrackReference` (I–252)

Removes a track reference from a track.

`SetTrackReference` (III–1665)

Modifies an existing track reference.

`GetTrackReference` (I–541)

Retrieves the track identifier contained in an existing track reference.

`GetNextTrackReferenceType` (I–503)

Determines all of the track reference types that are defined for a given track.

`GetTrackReferenceCount` (I–542)

Determines how many track references of a given type exist for a track.

Working With Alternate Tracks

The Movie Toolbox allows you to define alternate tracks in a movie. You can use alternate tracks to support multiple languages or to present different levels of visual quality in the movie.

You collect alternate tracks into groups. Alternate track groups are collections of tracks that conceptually represent some data but are appropriate for use in different play environments. The Movie Toolbox selects one track from each alternate group when it plays the movie. For example, you could create a movie that has three separate audio tracks: one in English, one in French, and one in Spanish. You would collect these audio tracks into an alternate group. When the user plays the movie, the Movie Toolbox selects the track from this group that corresponds to the current language setting for the movie.

Similarly, you can use alternate tracks to store data of different quality. When the

user plays the movie, the Movie Toolbox selects the track that best suits the capabilities of the Macintosh computer on which the movie is being played. In this manner, you can create a single movie that can accommodate the playback characteristics of a number of different computer configurations.

You set a movie's language by calling `SetMovieLanguage` (III–1620). To establish alternate groups of tracks, you can use `SetTrackAlternate` (III–1656) and `GetTrackAlternate` (I–522). You can work with the language and quality characteristics of media by calling `GetMediaLanguage` (I–436), `SetMediaLanguage`, `GetMediaQuality` (I–441), and `SetMediaQuality` (III–1605).

By default, the Movie Toolbox automatically selects the appropriate tracks to play according to a movie's quality and language settings, as well as the capabilities of the Macintosh computer. Whenever your application calls `SetMovieGWorld` (III–1619), `SetMovieBox`, `UpdateMovie` (III–1981), or `SetMovieMatrix` (III–1622), the Movie Toolbox checks each alternate group for an appropriate track. However, you can control this selection process. Use `SetAutoTrackAlternatesEnabled` (III–1570) to enable or disable automatic track selection. `SelectMovieAlternates` (III–1569) instructs the Movie Toolbox to select appropriate tracks immediately. If no tracks in an alternate track group are enabled, then the Movie Toolbox does not activate any track from that group during automatic track selection.

Functions

`SetMovieLanguage` (III–1620)

Specifies a movie's localized language or **region code**.

`SetTrackAlternate` (III–1656)

Adds tracks to, or remove tracks from, alternate groups.

`GetTrackAlternate` (I–522)

Determines all the tracks in an alternate group.

`GetMediaLanguage` (I–436)

Returns a media's localized language or **region code**.

`SetMediaLanguage` (III–1600)

Sets a media's localized language or **region code**.

`GetMediaQuality` (I–441)

Returns a media's quality level value.

`SetMediaQuality` (III–1605)

Sets a media's quality level value.

`SetAutoTrackAlternatesEnabled` (III–1570)

Enables or disables automatic track selection by the Movie Toolbox.

`SelectMovieAlternates` (III–1569)

Instructs the Movie Toolbox to select appropriate tracks immediately.

See Also

For language and geographic region identifiers, see “Localization Codes” (IV–2673).

Working With Track Sound

Two functions operate on the static 3D sound setting for a track. By constantly setting the value, it is possible for an application to make a track’s sound move in 3D space. If it is necessary to store dynamically changing 3D sound settings for the track, this can be done using the modifier track mechanism in conjunction with a **tween** track.

These functions are listed below.

Functions

`SetTrackSoundLocalizationSettings` (III–1666)

Applies 3D sound effect data to a track.

`GetTrackSoundLocalizationSettings` (I–543)

Returns a handle to a copy of the current 3D sound settings for a specified track.

Working With Data References

Media structures identify how and where to find their sample data by means of data references. For sound and video media, data references identify files, memory areas, or net addresses that contain media data. Media handlers use these data references in order to manipulate media data. A single media may contain one or more data references. The Movie Toolbox identifies a media’s data references with an index value. Index values always range from 1 to the number of references in the media. Data reference indexes provide a convenient way to access each reference in a media.

You can use `GetMediaDataRef` (I–427) to retrieve information about a media’s data reference. You can add a data reference to a media by calling `AddMediaDataRef` (I–26).

Your application can determine the number of data references in a media by calling `GetMediaDataRefCount` (I–428).

Functions

`GetMediaDataRef` (I–427)

Returns a copy of a specified data reference.

`AddMediaDataRef` (I–26)

Adds a data reference to a media.

`GetMediaDataRefCount` (I–428)

Determines the number of data references in a media.

Determining Movie Creation and Modification Time

The Movie Toolbox maintains two timestamps in every movie, track, and media. One timestamp, the creation date, indicates the date and time when the item was created. The other, the modification date, contains the date and time when the item was last changed and saved. The timestamp value is in the same format as Macintosh file system creation and modification times; that is, the timestamp indicates the number of seconds since midnight, January 1, 1904.

You can use `GetMovieCreationTime` (I–465) and `GetMovieModificationTime` to work with movie creation and modification dates. You can use `GetTrackCreationTime` (I–524) and `GetTrackModificationTime` to retrieve a track's creation and modification dates. You can use `GetMediaCreationTime` (I–424) and `GetMediaModificationTime` to get a media's creation and modification dates.

Functions

`GetMovieCreationTime` (I–465)

Returns the movie's creation date and time information.

`GetMovieModificationTime` (I–479)

Returns a movie's modification date and time.

`GetTrackCreationTime` (I–524)

Returns a track's creation date and time.

`GetTrackModificationTime` (I–536)

Returns a track's modification date and time.

`GetMediaCreationTime` (I–424)

Returns the creation date and time stored in a media.

`GetMediaModificationTime` (I–437)

Returns a media's modification date and time.

Working With Media Samples

The Movie Toolbox provides a number of functions that allow applications to determine information about a movie's sample data.

Your application can use `GetMovieDataSize` (I–465), `GetTrackDataSize`, and `GetMediaDataSize` (I–429) to determine the size, in bytes, of the data stored in a media, movie, or track.

You can use `GetMediaSampleDescriptionCount` (I–447) and `GetMediaSampleDescription` to retrieve a media's sample descriptions. `SetMediaSampleDescription` (III–1606) enables you to change the contents of a particular sample description associated with a media. `GetMediaSampleCount` (I–445) determines the number of samples in a media. `SampleNumToMediaTime` (III–1526) and `MediaTimeToSampleNum` allow you to convert from a time value to a sample number and vice versa.

Functions

`GetMovieDataSize` (I–465)

Determines the size of the sample data in a segment of a movie.

`GetTrackDataSize` (I–525)

Determines the size, in bytes, of the sample data in a segment of a track.

`GetMediaDataSize` (I–429)

Determines the size, in bytes, of the sample data in a media segment.

`GetMediaSampleDescriptionCount` (I–447)

Returns the number of sample descriptions in a media.

`GetMediaSampleDescription` (I–445)

Retrieves a `SampleDescription` (IV–2414) structure from a media.

`SetMediaSampleDescription` (III–1606)

Changes the contents of a particular `SampleDescription` (IV–2414) structure of a specified media.

`GetMediaSampleCount` (I–445)

Determines the number of samples in a media.

`SampleNumToMediaTime` (III–1526)

Finds the time at which a specified sample plays.

`MediaTimeToSampleNum` (II–913)

Lets you find the sample that contains the data for a specified time.

See Also

You can use the functions described in “Finding Interesting Times” (V–2735) to locate specific samples in a media. Also see “Adding Samples to Media Structures” (V–2752) for information about functions that allow you to retrieve sample data from a media.

Working With Movie User Data

Each movie, track, and media can contain a **user data list**, which your application can use in any way you want. A user data list contains all the user data for a movie, track, or media. Each user data list may contain one or more user data items.

All QuickTime user data items share several attributes. First, each user data item carries a type identifier, stored in a long integer. Apple has reserved all lowercase user data type values. You are free to create user data type values using uppercase or mixed case. Apple recommends using type values that begin with the © character (ASCII 0xA9) to specify user data items that store text data. User data items of these types must contain text data only.

Second, the Movie Toolbox allows you to create more than one user data item in a user data list. Therefore, each user data item is identified by a unique index. Index values are assigned sequentially within a user data type and start at 1.

Finally, you may create alternate text for a given user data text item. For example, you may want to support multiple languages and may therefore want to create different text for each language. The Movie Toolbox allows you to specify different versions of the text of a single user data item. These versions are distinguished by their **region code** values.

Before you can work with the contents of a **user data list**, you must obtain a reference to the list. `GetMovieUserData` (I–499), `GetTrackUserData`, and `GetMediaUserData` (I–456) allow you to get a reference to a user data list. You can then use `GetUserData` (I–547), `AddUserData` (I–44), and `RemoveUserData` (III–1459) to

work with the items contained in the user data list. If your user data items contain text data, you can use `AddUserDataText` (I–45), `GetUserDataText`, and `RemoveUserDataText` (III–1460) to work with the text of a user data item. Note that a single user data item can store either text or other data, but not both.

You can count the number of user data items of a specified type in a movie, track, or media by calling `CountUserDataTypes` (I–144). You can use `GetNextUserDataTypes` (I–503) to scan all the types of user data in a specified user data list.

The Movie Toolbox also supplies a number of functions for the manipulation of user data. `SetUserDataItem` (III–1673) and `GetUserDataItem` (I–548) allow easy access to data stored in user data items. `NewUserData` (II–1124) and `DisposeUserData` (I–303) provide for the use of user data outside of the immediate context of QuickTime movies.

Your applications and components can also create user data structures. `PutUserDataIntoHandle` (II–1154) and the `NewUserDataFromHandle` (II–1124) permit user data to be stored and retrieved in atoms.

Functions

`GetMovieUserData` (I–499)

Obtains access to a movie's **user data list**.

`GetTrackUserData` (I–546)

Obtains access to a track's **user data list**.

`GetMediaUserData` (I–456)

Obtains access to a media's **user data list**.

`GetUserData` (I–547)

Returns a specified user data item.

`AddUserData` (I–44)

Adds an item to a **user data list**.

`RemoveUserData` (III–1459)

Removes an item from a **user data list**.

`GetUserDataText` (I–549)

Retrieves language-tagged text from an item in a **user data list**.

`AddUserDataText` (I–45)

Places language-tagged text into an item in a **user data list**.

`RemoveUserDataText` (III–1460)

Removes language-tagged text from an item in a **user data list**.

`CountUserDataTypes` (I–144)

Determines the number of items of a given type in a **user data list**.

`GetNextUserDataTypes` (I–503)

Retrieves the next user data type in a specified **user data list**.

`SetUserDataItem` (III–1673)

Sets an item in a **user data list**.

`GetUserDataItem` (I–548)

Returns a specified user data item.

`NewUserData` (II–1124)

Creates a new user data structure.

`DisposeUserData` (I–303)

Disposes of a user data structure created by `NewUserData` (II–1124).

`PutUserDataIntoHandle` (II–1154)

Takes a specified user data structure and replaces the contents of the handle with a publicly parsable form of the user data.

`NewUserDataFromHandle` (II–1124)

Creates a new user data structure from a handle.

Managing the Video Frame Playback Rate

Two functions let you determine the rate at which a QuickTime movie plays back each video frame. You should use these functions for debugging.

These functions are listed below.

Functions

`VideoMediaGetStatistics` (III–2059)

Returns the play-back frame rate of a movie.

`VideoMediaResetStatistics` (III–2059)

Resets the video media handler's counters before using `VideoMediaGetStatistics` (III–2059) to determine the frame rate of a movie.

Editing Movies

The Movie Toolbox provides a number of functions that allow applications to edit existing movies or create the contents of new movies.

High-Level Movie Editing Functions

The Movie Toolbox provides a set of high-level functions that allow you to edit movies. These functions work with a movie's current selection, defined by a starting time and a duration. If you are calling the Movie Toolbox directly to do editing, you should use these functions.

The movies created by these functions contain references to the data in the source movie. Because the new movies contain references and not data, they are small and easily moved to and from the scrap. If you delete the movie that contains the data, the data references in the new movies are no longer valid and the new movies cannot be played. Therefore, before you delete the original movie, you should call `FlattenMovie` (I-372) for each of the new movies. This function copies the data into each of the new movies, eliminating the data references.

Note that the Movie Toolbox does not always copy empty tracks from the source movie to the movies that are created by these functions. Specifically, the Movie Toolbox preserves the empty tracks until you paste or add the selection into the destination movie. At that time, the Movie Toolbox removes the empty tracks from the selection. In addition, if a track in the source movie has trailing empty space, the Movie Toolbox removes that empty space from the track when it is copied into the new movie. Therefore, if you want to add a segment beyond the end of a movie, you insert the space when you insert the new segment using `InsertMovieSegment` (II-762).

The Movie Toolbox allows you to paste different data types into a movie. For example, `QuickDraw` pictures and standard sound data can be pasted directly into a movie. If you are using the movie controller component, you do not need to use these functions to paste different data types into a movie.

To get and change a movie's current selection, your application can call `GetMovieSelection` (I-491) and `SetMovieSelection` (III-1632). Your application can work with a movie's current selection by calling `CutMovieSelection` (I-166),

CopyMovieSelection, PasteMovieSelection (II-1139), ClearMovieSelection, and AddMovieSelection (I-38).

PutMovieOnScrap and NewMovieFromScrap (II-1085) enable your application to work with movies that are on the scrap. IsScrapMovie (II-775) examines the system scrap to determine whether it can translate any of the data into a movie.

PasteHandleIntoMovie (II-1137) takes the contents of a specified handle, together with its type, and pastes it into a movie. PutMovieIntoTypedHandle (II-1152) takes a movie (or a single track from within a movie) and converts it into a handle.

Functions

GetMovieSelection (I-491)

Returns information about a movie's current selection.

SetMovieSelection (III-1632)

Sets a movie's current selection.

CutMovieSelection (I-166)

Creates a new movie that contains the original movie's current selection.

CopyMovieSelection (I-139)

Creates a new movie that contains the original movie's current selection.

PasteMovieSelection (II-1139)

Places the tracks from one movie into another movie.

ClearMovieSelection (I-90)

Removes the segment of the movie that is defined by the current selection.

AddMovieSelection (I-38)

Adds one or more tracks to a movie.

PutMovieOnScrap (II-1153)

Places a movie into the Macintosh scrap.

NewMovieFromScrap (II-1085)

Creates a movie from the contents of the scrap.

IsScrapMovie (II-775)

Checks the system scrap to find out if it can translate any of the data into a movie.

PasteHandleIntoMovie (II–1137)

Takes the contents of a specified handle, together with its type, and pastes it into a specified movie.

PutMovieIntoTypedHandle (II–1152)

Takes a movie, or a single track from within that movie, and converts it into a handle of a specified type.

See Also

See “Low-Level Movie Editing Functions” (V–2749).

Undo for Movies

The Movie Toolbox provides functions that allow you to save and restore the edit state of a movie. An edit state contains information that completely defines a movie’s content at the time you create the edit state. It is, in essence, a checkpoint in the edit session. You can manage a movie’s edit states in order to implement an undo capability for editing movies. For example, you can save a movie’s edit state before performing an editing operation, such as a cut, and later restore the old state. You can have several movie edit states obtained at different times during an editing session, and restore to any one of them at any time. In this manner, you can provide a multilevel undo capability.

Note that a movie’s edit state does not save everything about a movie. Most important, the edit state does not contain information about the movie’s spatial characteristics. For example, the edit state does not store the current boundary rectangle or clipping region. Consequently, edit states are best suited to supporting undo operations involving movie content, including track creation and removal. You can use other Movie Toolbox functions to support undo operations for movie characteristics.

You can use `NewMovieEditState` (II–1073) to save a movie’s edit state. Use `UseMovieEditState` (III–1984) to restore the movie to its condition according to a previous edit state. Your application must dispose of an edit state by calling `DisposeMovieEditState` (I–273). You must dispose of a movie’s edit states before you dispose of the movie.

Functions

`NewMovieEditState` (II–1073)

Creates an edit state.

UseMovieEditState (III-1984)

Returns a movie to the condition determined by an edit state created previously.

DisposeMovieEditState (I-273)

Disposes of an edit state.

Low-Level Movie Editing Functions

The Movie Toolbox provides a number of functions that allow your application to perform low-level editing operations on movies. These functions work with movie segments (pieces of a movie that are defined by a starting time and duration) and therefore give you a great deal of control over the editing process. These functions never copy the movie data; rather, they work with references to the movie's data.

Use CopyMovieSettings (I-140) to copy certain important settings from one movie to another. Use InsertMovieSegment (II-762) to copy a segment from one movie to another. Use InsertEmptyMovieSegment (II-759) to insert an empty segment into a movie. Use DeleteMovieSegment (I-250) to delete a segment from a movie.

You can change a segment's duration by calling the ScaleMovieSegment (III-1528) function. This function stretches or shrinks the segment to accommodate a specified duration.

Functions

CopyMovieSettings (I-140)

Copies many settings from one movie to another, overwriting the destination settings in the process.

InsertMovieSegment (II-762)

Copies part of one movie to another.

InsertEmptyMovieSegment (II-759)

Adds an empty segment to a movie.

DeleteMovieSegment (I-250)

Removes a specified segment from a movie.

ScaleMovieSegment (III-1528)

Changes the duration of a segment of a movie.

See Also

See “High-Level Movie Editing Functions” (V–2746). These functions let edit movies by working with the current selection.

Editing Tracks

The Movie Toolbox provides a number of functions that allow your application to perform editing operations on tracks. They work with track segments (pieces of a track that are defined by a starting time and duration) and therefore give you a great deal of control over the editing process. These functions may copy movie data, if required by the operation. Note that when you edit a track you may change the duration of the movie that contains that track.

`CopyTrackSettings` (I–141) lets you copy certain important settings from one track to another. You can use `InsertTrackSegment` (II–764) to copy a segment from one track to another (by reference or by moving data) or to copy a segment within a track. `InsertEmptyTrackSegment` (II–760) allows you to insert an empty segment into a track. You can use the `InsertMediaIntoTrack` (II–761) function to insert a media into a track.

You can delete a segment from a track by calling the `DeleteTrackSegment` (I–252) function. You can change a segment’s duration by calling the `ScaleTrackSegment` (III–1529) function. This function stretches or shrinks the segment to accommodate a specified duration.

You can use the `GetTrackEditRate` (I–530) function to determine the rate of the track edit of a specified track at an indicated time.

Functions

`CopyTrackSettings` (I–141)

Copies many settings from one track to another, overwriting the destination settings.

`InsertTrackSegment` (II–764)

Copies data into a track.

`InsertEmptyTrackSegment` (II–760)

Adds an empty segment to a track.

`InsertMediaIntoTrack` (II–761)

Inserts a reference to a media segment into a track.

DeleteTrackSegment (I–252)

Removes a specified segment from a track.

ScaleTrackSegment (III–1529)

Changes the duration of a segment of a track.

GetTrackEditRate (I–530)

Returns the rate of the track edit of a specified track at an indicated time.

AddEmptyTrackToMovie (I–23)

Duplicates a track from a movie into the same movie or into another movie.

Undo for Tracks

The Movie Toolbox provides functions that allow you to capture and restore the edit state of a track. As with the functions that manipulate a movie's edit state, you can manage a track's edit states in order to implement an undo capability for track editing. For example, you can capture a track's edit state before performing an editing operation, such as a cut, and later restore the old state. You can have several track edit states obtained at different times during an editing session, and you can restore to any one of them at any time. In this manner, you can provide a multilevel undo capability.

Note that a track's edit state does not save everything about the track. Most important, the edit state does not contain information about track spatial characteristics. For example, the edit state does not store the current clipping region. Consequently, edit states are best suited to supporting undo operations involving track content. You can use other Movie Toolbox functions to support undo operations for track characteristics.

Use `NewTrackEditState` (II–1119) to capture a track's edit state. Use `UseTrackEditState` (III–1984) to restore the track to its condition according to a previous edit state. You can dispose of an edit state by calling `DisposeTrackEditState` (I–300).

Functions

`NewTrackEditState` (II–1119)

Creates a new edit state for a given track.

`UseTrackEditState` (III–1984)

Returns a track to the condition determined by an edit state created previously.

DisposeTrackEditState (I-300)

Disposes of a movie's track edit state.

Adding Samples to Media Structures

Certain Movie Toolbox functions directly manipulate media samples. These functions are used only by applications that create movies or add data to existing movies.

You add samples to a media by calling `AddMediaSample` (I-27). You can indicate that the sample to be added is or is not a **sync sample**. You can obtain the data in a media sample by calling `GetMediaSample` (I-443). If you are going to add samples to a media, you must do so within a media-editing session. You start a media-editing session by calling `BeginMediaEdits` (I-56). Once you have finished adding samples to the media, you end the editing session by calling `EndMediaEdits` (I-335).

Once you have added samples to a media, you can work with references to those samples by calling `AddMediaSampleReference` (I-31) and `GetMediaSampleReference`. You do not have to be in a media-editing session to use these functions

Functions

`AddMediaSample` (I-27)

Adds sample data and a description to a media.

`GetMediaSample` (I-443)

Returns a sample from a movie data file.

`BeginMediaEdits` (I-56)

Starts a media-editing session.

`EndMediaEdits` (I-335)

Ends a media-editing session.

`AddMediaSampleReference` (I-31)

Works with samples that have already been added to a movie data file.

`AddMediaSampleReferences` (I-33)

Adds groups of samples to a movie data file.

`GetMediaSampleReference` (I-447)

Obtains reference information about samples that are stored in a movie data file.

`GetMediaSampleReferences` (I–449)

Obtains reference information about groups of samples that are stored in a movie.

`SetMediaDefaultDataRefIndex` (III–1597)

Specifies which of a media's data references is to be accessed during an editing session.

`SetMediaPreferredChunkSize` (III–1603)

Specifies a maximum chunk size for a media.

`GetMediaPreferredChunkSize` (I–440)

Retrieves the maximum chunk size for a media.

Manipulating Media Input Maps

The Movie Toolbox contains two functions for maintaining media input maps.

Each track has particular attributes such as size, position, and volume associated with it. The media input map of that track describes where the variable parameters are stored so that modifier tracks know where to send their data. When a track is copied, its input map is also copied.

Functions

`GetMediaInputMap` (I–435)

Returns a copy of the input map associated with a specified media.

`SetMediaInputMap` (III–1599)

Replaces the media's existing input map with a given input map.

Interacting With Media Handlers

The Movie Toolbox does not contain any support for specific media types. Rather, it delegates this work to media handler components. The Movie Toolbox provides a number of functions that allow your application to interact with media handlers.

Selecting Media Handlers

Media handler components are responsible for interpreting and manipulating a media's sample data. Each type of media has its own media handler, which deals with the specific characteristics of the media data. The Movie Toolbox provides a set of functions that allow you to gather information about a media handler and assign a particular media handler to a media.

Each media handler has an associated data handler for each data reference. The data handler is responsible for fetching, storing, and caching the data that the media handler uses. The Movie Toolbox provides functions that allow you to get information about data handlers and to assign a particular data handler to a media.

`GetMediaHandler` (I-432) and `GetMediaHandlerDescription` (I-432) allow you to retrieve information about a media handler. You can use the `SetMediaHandler` (III-1598) to assign a media handler to a media.

`GetMediaDataHandler` (I-425) and `GetMediaDataHandlerDescription` enable you to retrieve information about a data handler. Use `SetMediaDataHandler` (III-1594) to assign a data handler to a media.

Functions

`GetMediaHandler` (I-432)

Obtains a reference to a media handler component.

`GetDataHandler` (I-407)

Retrieves the best data handler component to use with a given data reference.

`GetMediaHandlerDescription` (I-432)

Retrieves information about a media handler.

`SetMediaHandler` (III-1598)

Assigns a specific media handler to a track.

`GetMediaDataHandler` (I-425)

Determines a media's data handler.

`GetMediaDataHandlerDescription` (I-426)

Retrieves information about a media's data handler.

`SetMediaDataHandler` (III-1594)

Assigns a data handler to a media.

Working With Media Handler Properties

Two functions are provided for setting and retrieving the property atom container of a media handler.

These functions are listed below.

Functions

`GetMediaPropertyAtom` (I–440)

Retrieves the property atom container of a media handler.

`SetMediaPropertyAtom` (III–1604)

Sets the property atom container of a media handler.

Video Media Handler Functions

Video media handlers are responsible for interpreting and manipulating video data. You can call them directly to work with graphics modes and color values that affect the display of video data.

Use `MediaSetGraphicsMode` (II–892) and `MediaGetGraphicsMode` to work with these characteristics.

Functions

`MediaSetGraphicsMode` (II–892)

Sets the graphics mode and blend color of any media handler.

`MediaGetGraphicsMode` (II–852)

Obtains the graphics mode and blend color values currently in use by any media handler.

Sound Media Handler Functions

Sound media handlers are responsible for interpreting and manipulating sound data. You can call them directly to work with their stereo balance settings.

Use `MediaSetSoundBalance` (II–904) and `MediaGetSoundBalance` to work with a handler's balance setting.

Functions

`MediaSetSoundBalance` (II–904)

Sets the sound balance of the media handler.

`MediaGetSoundBalance` (II–860)

Obtains the sound balance of the media handler.

Text Media Handler Functions

You can use text media handlers to add plain or styled text samples to a movie, indicate scrolling and highlighting properties for the text, search for text, and highlight specified text. A particular text sample has a default font, size, typeface, and color as well as a location (text box) within the track bounds to be drawn. The data format allows you to include style run information for the text. You can set flags to clip the display to the text box, inhibit automatic scaling of text as the track bounds are scaled, scroll the text, and specify if text is to be displayed at all.

You can use `TextMediaAddTextSample` (III–1936) to add text to a media.

`TextMediaAddTESample` (III–1933) allows you to specify a `TextEdit` handle (which may have multiple style runs) to be added to a media. `TextMediaAddHiliteSample` (III–1932) allows you to indicate highlighting for text that has just been added.

These functions convert text into the text media format and add it to a media. To use them, you need to create a text track and media, call `BeginMediaEdits` (I–56), call the appropriate text media handler function, call `EndMediaEdits` (I–335), and call `InsertMediaIntoTrack` (II–761).

The format of the text data that is added to the media is a 16-bit length word followed by the text. The length word specifies the number of bytes in the text. Optionally, one or more atoms of additional data may follow.

The Movie Toolbox provides functions that allow you to search for and highlight text. You can use `TextMediaFindNextText` (III–1941) to search for text in a text track and `TextMediaHiliteTextSample` (III–1944) to highlight specified text in a text track. You can use `TextMediaSetTextProc` (III–1947) to specify a callback to be invoked whenever a new text sample is added to a movie.

Functions

`TextMediaAddTextSample` (III–1936)

Adds a single block of styled text to an existing media.

`TextMediaAddTESample` (III–1933)

Specifies a `TextEdit` handle to be added to a specified media.

`TextMediaAddHiliteSample` (III–1932)

Provides dynamic highlighting of text.

`TextMediaFindNextText` (III–1941)

Searches for text with a specified media handler starting at a given time.

`TextMediaHiliteTextSample` (III–1944)

Specifies selected text to be highlighted for a given text media handler.

`TextMediaSetTextSampleData` (III–1950)

Sets values before calling `TextMediaAddTextSample` (III–1936) or `TextMediaAddTESample` (III–1933).

`TextMediaSetTextProc` (III–1947)

Specifies a custom function to be called whenever a text sample is displayed in a movie.

Callback

`TextMediaProc` (III–2150)

A callback that can be called whenever a text sample is displayed in a movie.

See Also

For more information on styled text, style runs, and `TextEdit`, see *Inside Macintosh: Text* (listed in the **bibliography**).

Previewing Files

File **previews** contain information that gives the user an idea of a file's contents without opening the file. Typically, a file's preview is a small PICT image (called a thumbnail), but previews may also contain other types of information that is appropriate to the type of file being considered. For example, a text file's preview might tell the user when the file was created and what it discusses.

Creating File Previews

The Movie Toolbox provides two functions that allow you to create file **previews**.

You can use `MakeFilePreview` (II–784) to create a **preview** for a file. `AddFilePreview` (I–24) allows you to add the preview to a file.

Functions

`MakeFilePreview` (II–784)

Creates a **preview** for a file.

`AddFilePreview` (I–24)

Adds a **preview** to a file.

Displaying File Previews

Four Movie Toolbox functions let you display file **previews**.

Two functions allow you to display file **previews** in an Open dialog box in System 6 using standard file reply structures: `SFGetFilePreview` (III–1674) and `SFPGetFilePreview` (III–1676). Two newer functions allow you to display file previews in an Open dialog box in System 7 using standard file reply structures: `StandardGetFilePreview` (III–1910) and `CustomGetFilePreview`.

These functions allow the user to automatically convert files to movies if your application requests movies. If there is a file that can be converted into a movie file using a movie import component, then the file is shown in the Standard File dialog box in addition to any movies. When the user selects the file, the Open button changes to a Convert button. Choosing Convert displays a dialog box that allows the user to choose where the converted file should be saved.

When conversion is complete, the converted file is returned to the calling application as the movie that the user chose. If you want to disable automatic file conversion in your application, you must write a file filter function and pass it to the file preview display function you are using.

Functions

`SFGetFilePreview` (III–1674)

Displays a file **preview** in an Open dialog box using a standard file reply structure.

`SFPGetFilePreview` (III–1676)

Displays file **previews** in an Open dialog box using a standard file reply structure.

`StandardGetFilePreview` (III–1910)

Displays file **previews** in an Open dialog box using a standard file reply structure.

`CustomGetFilePreview` (I–163)

Presents an Open dialog box to the user and allows the user to view file **previews**.

Working With Time Bases

The Movie Toolbox provides a number of functions that allow you to work with time bases. A `QuickTime` time base defines the time coordinate system of a movie. However, you can also use `QuickTime` time bases to provide general timing services.

Note that time base functions do not change the value of the Movie Toolbox **sticky error** value. Although most time base functions can be used at interrupt time, several of the Movie Toolbox functions cannot. For further information, see the detailed function descriptions.

Creating and Disposing of Time Bases

The Movie Toolbox provides several functions your application can use to create and dispose of time bases.

`NewTimeBase` (II–1119) lets you create a new time base. You can use `DisposeTimeBase` (I–299) to dispose of a time base once you are finished with it.

Time bases rely on either a clock component or another time base for their time source. You can use `SetTimeBaseMasterTimeBase` (III–1650) to cause one time base to be based on another time base. `GetTimeBaseMasterTimeBase` (I–517) allows you to determine the master time base of a given time base.

You can assign a clock component to a time base; that clock then acts as the master clock for the time base. You can use `SetTimeBaseMasterClock` (III–1649) to assign a clock component to a time base. `GetTimeBaseMasterClock` (I–516) enables you to determine the clock component that is assigned to a time base.

You can change the offset between a time base and its time source by calling `SetTimeBaseZero` (III–1655). You can set the time source of a movie by calling `SetMovieMasterTimeBase` (III–1621) or `SetMovieMasterClock`.

Functions

`NewTimeBase` (II–1119)

Obtains a new time base.

`DisposeTimeBase` (I–299)

Disposes of a time base once you are finished with it.

`SetTimeBaseMasterTimeBase` (III–1650)

Assigns a master time base to a time base.

`GetTimeBaseMasterTimeBase` (I–517)

Determines the master time base that is assigned to a time base.

`SetTimeBaseMasterClock` (III–1649)

Assigns a clock component to a time base.

`GetTimeBaseMasterClock` (I–516)

Determines the clock component that is assigned to a time base.

`SetTimeBaseZero` (III–1655)

Changes the offset from a time base to either its master time base or its clock component.

`SetMovieMasterTimeBase` (III–1621)

Assigns a master time base to a movie.

`SetMovieMasterClock` (III–1620)

Assigns a clock component to a movie.

Working With Time Base Values

Every time base contains a rate, a start time, a stop time, a current time, and some status information. The Movie Toolbox provides a number of functions that allow your application to work with the contents of a time base.

`GetTimeBaseTime` (I–521) lets you retrieve the current time value of a time base. You can set the current time value by calling `SetTimeBaseTime` (III–1653); this function requires you to provide a **time structure**. Alternatively, you can set the current time

based on a time value by calling `SetTimeBaseValue` (III–1654).

You can determine the rate of a time base by calling `GetTimeBaseRate` (I–518). You can set the rate of a time base by calling `SetTimeBaseRate` (III–1651). You can determine the effective rate of a specified time base (relative to the master time base to which it is subordinate) by calling `GetTimeBaseEffectiveRate` (I–514).

You can retrieve the start time of a time base by calling `GetTimeBaseStartTime` (I–518). You can set the start time of a time base by calling `SetTimeBaseStartTime` (III–1652). Similarly, you can use `GetTimeBaseStopTime` (I–520) and `SetTimeBaseStopTime` to work with the stop time of a time base.

The Movie Toolbox provides functions that allow you to work with the status information of a time base. `GetTimeBaseStatus` (I–519) allows you to read the current status of a time base. `GetTimeBaseFlags` (I–515) helps you obtain the control flags of a time base. You can set these flags by calling `SetTimeBaseFlags` (III–1648).

Functions

`GetTimeBaseTime` (I–521)

Obtains the current time value from a time base.

`SetTimeBaseTime` (III–1653)

Sets the current time of a time base.

`SetTimeBaseValue` (III–1654)

Sets the current time of a time base.

`GetTimeBaseRate` (I–518)

Retrieves the rate of a time base.

`SetTimeBaseRate` (III–1651)

Sets the rate of a time base.

`GetTimeBaseEffectiveRate` (I–514)

Returns the effective rate at which the specified time base is moving relative to its master clock.

`GetTimeBaseStartTime` (I–518)

Determines the start time of a time base.

`SetTimeBaseStartTime` (III–1652)

Sets the start time of a time base.

GetTimeBaseStopTime (I-520)

Determines the stop time of a time base.

SetTimeBaseStopTime (III-1652)

Sets the stop time of a time base.

GetTimeBaseStatus (I-519)

Determines when the current time of a time base would fall outside of the range of values specified by the time base's start and stop times.

GetTimeBaseFlags (I-515)

Obtains the control flags of a time base.

SetTimeBaseFlags (III-1648)

Sets the contents of the control flags of a time base.

Working With Times

The Movie Toolbox provides a number of functions that allow you to work with **time structures**; see “The Time Structure” (V-2710). You can use time structures to represent either time values or durations. Time values specify a point in time, relative to a given time base. Durations specify a span of time, relative to a given time scale. Durations are represented by time structures that have the time base set to 0 (that is, the base field in the time structure is set to NIL).

You can use `ConvertTime` (I-137) to convert a time you obtain from one time base into a time that is relative to another time base. Similarly, you can use `ConvertTimeScale` (I-138) to convert a time from one time scale to another. You can add two times by calling `AddTime` (I-41); you can subtract two times with `SubtractTime` (III-1915).

Functions

`ConvertTime` (I-137)

Converts a time obtained from one time base into a time that is relative to another time base.

`ConvertTimeScale` (I-138)

Converts a time from one time scale into a time that is relative to another time scale.

`AddTime` (I-41)

Adds two times.

SubtractTime (III-1915)

Subtracts one time from another.

Time Base Callback Functions

If your application uses QuickTime time bases, it may define callback functions that are associated with a specific time base. Your application can then use these callback functions to perform activities that are triggered by temporal events, such as a certain time being reached or a specified rate being achieved. The time base functions of the Movie Toolbox interact with clock components to schedule the invocation of these callback functions; the clock components are responsible for invoking the callback function at its scheduled time.

You can define three types of **callback events**, distinguished by the nature of the temporal event that triggers the Movie Toolbox to call your function. They include events that are triggered at a specified time, events that are triggered when the rate reaches a specified value, and events that are triggered when the time value of a time base changes by an amount different from the time base's rate.

You specify a **callback event**'s type when you define the callback event, using `NewCallback` (II-1053). This function also allocates the memory to support the callback event.

You also specify whether your event can occur at interrupt time. Your function is called closer to the triggering event at interrupt time, but it is subject to all the restrictions of interrupt functions (for example, your callback function cannot cause memory to be moved). If your function is not called at interrupt time, you are free of these restrictions, but your function may be called later because the invocation is delayed to avoid interrupt time.

You schedule a callback event by calling `CallMeWhen` (I-67). Call `CancelCallback` (I-70) to unschedule a callback event. When you are done with a callback event, you dispose of it by calling `DisposeCallback` (I-256).

You can retrieve the time base of a callback event by calling `GetCallbackTimeBase` (I-384). You can obtain the type of a callback event by calling `GetCallbackType` (I-384).

Functions

NewCallback (II–1053)

Creates a new **callback event**.

CallMeWhen (I–67)

Schedules a **callback event**.

CancelCallback (I–70)

Cancels a **callback event** before it executes.

DisposeCallback (I–256)

Disposes of a **callback event**.

GetCallbackTimeBase (I–384)

Retrieves the time base of a **callback event**.

GetCallbackType (I–384)

Retrieves a **callback event**'s type.

Callbacks

QTCallbackProc (III–2124)

A generic callback function, installed by CallMeWhen (I–67).

Working With Matrices

The Movie Toolbox makes extensive use of transformation matrices to define graphical operations that are performed on movies when they are displayed. A transformation matrix defines how to map points from one coordinate space into another coordinate space. By modifying the contents of a transformation matrix, you can perform several standard graphical display operations, including translation, rotation, and scaling. The Movie Toolbox provides a set of functions that make it easy for you to manipulate and apply translation matrices.

Managing Matrices

The Movie Toolbox provides several functions that help you create, modify, and compare matrices.

These functions are listed below.

Functions

`SetIdentityMatrix` (III–1593)

Sets the contents of a matrix so that it performs no transformation.

`GetMatrixType` (I–419)

Obtains information about a matrix.

`CopyMatrix` (I–139)

Copies the contents of one matrix into another matrix.

`EqualMatrix` (I–338)

Compares two matrices and returns a result that indicates whether the matrices are equal.

`TranslateMatrix` (III–1956)

Adds a translation value to a specified matrix.

`ScaleMatrix` (III–1527)

Modifies the contents of a matrix so that it defines a scaling operation.

`RotateMatrix` (III–1462)

Modifies the contents of a matrix so that it defines a rotation operation.

`SkewMatrix` (III–1827)

Modifies the contents of a matrix so that it defines a skew transformation.

`ConcatMatrix` (I–128)

Concatenates two matrices, combining the transformations described by both matrices into a single matrix.

`InverseMatrix` (II–774)

Creates a new matrix that is the inverse of a specified matrix.

`RectMatrix` (III–1451)

Creates a matrix that performs the translate and scale operation described by the relationship between two rectangles.

`MapMatrix` (II–794)

Alters an existing matrix so that it defines a transformation from one rectangle to another.

Applying Matrix Transformations

The Movie Toolbox provides several functions that help you apply matrices to various graphic entities.

These functions are listed below.

Functions

`TransformPoints` (III–1953)

Transforms a set of QuickDraw points through a specified matrix.

`TransformFixedPoints` (III–1951)

Transforms a set of fixed points through a specified matrix.

`TransformRect` (III–1954)

Transforms the upper-left and lower-right points of a rectangle through a specified matrix.

`TransformFixedRect` (III–1952)

Transforms the upper-left and lower-right points of a rectangle through a matrix that is specified by fixed points.

`TransformRgn` (III–1955)

Applies a specified matrix to a region.

Working With QT Atoms

QuickTime atoms are basic data containers used in QuickTime programming. QuickTime contains a complete set of functions for creating, manipulating, finding, and accessing QT atoms and atom containers.

Creating and Disposing of Atom Containers

The Movie Toolbox provides two calls to let you create and dispose of QuickTime atom containers.

Before you can add atoms to an atom container, you must first create the container by calling `QTNewAtomContainer` (II–1203). When you have finished using an atom container, you should dispose of it by calling `QTDisposeAtomContainer` (II–1164).

Functions

`QTNewAtomContainer` (II–1203)

Creates a new atom container.

`QTDisposeAtomContainer` (II–1164)

Disposes of an atom container.

Creating New Atoms

Once you have created an atom container, the Movie Toolbox provides a call to let you place new QuickTime atoms in it.

`QTInsertChild` (II–1183) creates a new child atom for a parent atom. The caller specifies an atom type and atom ID for the new atom. If you specify a value of 0 for the atom ID, `QTInsertChild` (II–1183) assigns a unique ID to the atom.

This function inserts the atom in the parent’s child list at the index specified by the `index` parameter; any existing atoms at the same index or greater are moved toward the end of the child list. If you specify a value of 0 for the `index` parameter, it inserts the atom at the end of the child list.

Functions

`QTInsertChild` (II–1183)

Creates a new child atom of the specified parent atom.

Copying Existing Atoms

The Movie Toolbox provides several functions for copying existing atoms within an atom container.

`QTInsertChildren` (II–1185) inserts a container of atoms as children of a parent atom in another atom container. `QTReplaceAtom` (II–1217) replaces an atom and its children with a different atom and its children. You can call `QTSwapAtoms` (II–1315) to swap the contents of two atoms in an atom container; after swapping, the ID and index of

each atom remains the same. `QTCopyAtom` (II–1158) copies an atom and its children to a new atom container.

Functions

`QTInsertChildren` (II–1185)

Inserts a container of atoms as children of the specified parent atom.

`QTReplaceAtom` (II–1217)

Replaces the contents of an atom and its children with a different atom and its children.

`QTSwapAtoms` (II–1315)

Swaps the contents of two atoms in an atom container.

`QTCopyAtom` (II–1158)

Copies an atom and its children to a new atom container.

Retrieving Atoms and Atom Data

The Movie Toolbox provides functions you can use to retrieve information about the types of a parent atom's children, to search for a specific atom, and to retrieve a leaf atom's data.

`QTCountChildrenOfType` (II–1160) returns the number of children of a given atom type for a parent atom. `QTGetNextChildType` (II–1178) returns the next atom type in the child list of a parent atom.

You call `QTFindChildByIndex` (II–1167) to search for and retrieve a parent atom's child by its type and index within that type. You can call `QTFindChildByID` (II–1166) to search for and retrieve a parent atom's child by its type and ID. Once you have retrieved a child atom, you can call `QTNextChildAnyType` (II–1209) to retrieve subsequent children of a parent atom. It returns an offset to the next atom of any type in a parent atom's child list. This function is useful for iterating through a parent atom's children quickly.

QuickTime also provides functions for retrieving an atom's type, ID, and data. You can call `QTGetAtomTypeAndID` (II–1172) to retrieve an atom's type and ID. You can access an atom's data in one of three ways. To copy an atom's data to a handle, you can use `QTCopyAtomDataToHandle` (II–1158).

To access an atom's data directly, you should lock the atom container in memory by

calling `QTLockContainer` (II–1187). Once the container is locked, you can call `QTGetAtomDataPtr` (II–1171) to retrieve a pointer to an atom’s data. When you have finished accessing the atom’s data, you should call the `QTUnlockContainer` (II–1316) function to unlock the container in memory.

Functions

`QTCountChildrenOfType` (II–1160)

Returns the number of atoms of a given type in the child list of the specified parent atom.

`QTGetNextChildType` (II–1178)

Returns the next atom type in the child list of the specified parent atom.

`QTFindChildByIndex` (II–1167)

Retrieves an atom by index from the child list of the specified parent atom.

`QTFindChildByID` (II–1166)

Retrieves an atom by ID from the child list of the specified parent atom.

`QTNextChildAnyType` (II–1209)

Returns the next atom in the child list of the specified parent atom.

`QTGetAtomTypeAndID` (II–1172)

Retrieves an atom’s type and ID.

`QTCopyAtomDataToHandle` (II–1158)

Copies the specified leaf atom’s data to a handle.

`QTCopyAtomDataToPtr` (II–1159)

Copies the specified leaf atom’s data to a buffer.

`QTGetAtomDataPtr` (II–1171)

Retrieves a pointer to the atom data for a specified leaf atom.

`QTLockContainer` (II–1187)

Locks an atom container in memory.

`QTUnlockContainer` (II–1316)

Unlocks an atom container in memory.

Modifying Atoms

QuickTime provides functions that you can call to modify attributes or data associated with an atom in an atom container.

To modify an atom's ID, call `QTSetAtomID` (II-1225). You can use `QTSetAtomData` (II-1223) to update the data associated with a leaf atom in an atom container.

Functions

`QTSetAtomID` (II-1225)

Changes the ID of an atom.

`QTSetAtomData` (II-1223)

Changes the data of a leaf atom.

Removing Atoms From an Atom Container

The Movie Toolbox provides two function that let you remove atoms from a container.

`QTRemoveAtom` (II-1215) removes an atom and its children, if any, from a container. `QTRemoveChildren` (II-1216) removes an atom's children from a container, but does not remove the atom itself. You can also use this function to remove all the atoms in an atom container. To do so, pass the constant `kParentAtomIsContainer` for the `atom` parameter.

Functions

`QTRemoveAtom` (II-1215)

Removes an atom and its children from the specified atom container.

`QTRemoveChildren` (II-1216)

Removes all the children of an atom from the specified atom container.

Working With Access Keys

Certain compression formats, such as the Intel Indeo format for video compression, allow data to be encrypted when it is compressed. For software to gain access to the encrypted data, the access key for the data must be registered with QuickTime. For example, a CD-ROM title would register one or more access keys for encrypted video data it presents. Once the access keys are registered, the keys are available to the decompressor component that will be used to decompress the data. The CD-ROM title can then use the encrypted data in the same way it uses unencrypted

data.

The scope of an access key is determined when it is registered. If a key is registered as a system access key, data protected by that key is available to any QuickTime client on the computer. If a key is registered as an application access key, data protected by the key is available only to QuickTime clients of the application that registers the key.

Although most access keys are character strings, an access key can be of any data type. Users can enter, edit, and delete system access keys in the QuickTime control panel. Access keys entered in the QuickTime control panel are always registered as system access keys.

Registering and Unregistering Access Keys

Two functions let your application register and unregister access keys.

Use `QTRegisterAccessKey` (II–1214) to register an access key; use `QTUnregisterAccessKey` (II–1317) to unregister it.

Functions

`QTRegisterAccessKey` (II–1214)

Registers an access key.

`QTUnregisterAccessKey` (II–1317)

Removes a previously registered access key.

Retrieving Access Keys

A Movie Toolbox function lets your application determine all the application and system access keys of a specified access key type.

Use `QTGetAccessKeys` (II–1168) to return an atom container with the current access keys.

Functions

`QTGetAccessKeys` (II–1168)

Returns all the application and system access keys of a specified access key type.

Managing Progressive Downloads

The Movie Toolbox includes support for progressive downloads, which allow part of a movie to be displayed before all of its data has been received over a network or other slow link. Applications that use the movie controller component provided by Apple automatically get support for progressive downloads.

High-Level Download Control

Applications that do not use the standard movie controller can use two high-level functions to control progressive downloads.

You can use `QTMovieNeedsTimeTable` (II-1202) and `GetMaxLoadedTimeInMovie` to determine whether a movie is being progressively downloaded and, if so, to see how much of it has already been downloaded.

Functions

`QTMovieNeedsTimeTable` (II-1202)

Returns whether a movie is being progressively downloaded.

`GetMaxLoadedTimeInMovie` (I-423)

When a movie is being progressively downloaded, returns the duration of the part of a movie that has already been downloaded.

See Also

See “Low-Level Download Control” (V-2772).

Low-Level Download Control

Some applications may need more control over progressive downloads, such as control over individual tracks or media, than is possible with the high-level functions for progressive downloads.

You can use one or both of the functions `MakeTrackTimeTable` (II-793) and `MakeMediaTimeTable` for low-level download control.

Functions

`MakeTrackTimeTable` (II-793)

Returns a time table for a specified track in a movie.

`MakeMediaTimeTable` (II-788)

Returns a time table for the specified media.

See Also

See “High-Level Download Control” (V-2772).

Using Movie Controllers

Movie controller components provide a high-level interface that allows your application to present movies to users quickly and easily. Movie controller components eliminate much of the complexity of working with movies by assuming primary responsibility for the movie, freeing your application to focus on the unique services it offers to users.

Movie Controller Actions

Movie controller actions are designated by integer constants (defined by the `mcAction` data type) used by movie controller components.

Applications that use movie controller components can invoke these actions by calling `MCDoAction` (II-801). If the application includes an action filter function, that function may receive any of these actions. You can install an action filter with `MCSetActionFilterWithRefCon` (II-829).

Your action filter function should refer any actions that you do not want to handle back to the calling movie controller component. If you use any Movie Toolbox functions that modify the movie in your action filter function, be sure to call `MCMovieChanged` (II-822).

Functions

`MCDoAction` (II-801)

Invokes a movie controller component and makes it perform a specified action.

`MCSetActionFilterWithRefCon` (II–829)

Establishes an action filter function for a movie controller.

`MCMovieChanged` (II–822)

Informs a movie controller component that your application has used the Movie Toolbox to change the characteristics of its associated movie.

Callback

`MCActionFilterWithRefConProc` (III–2097)

Responds to movie controller actions with a reference constant.

Associating Movies With Controllers

Once your application has established a connection to a movie controller component, you may associate one movie with a movie controller. By default, the new controller has editing and keystroke processing turned off.

You create a new movie controller and assign it to a movie by calling `NewMovieController` (II–1071). This is the easiest way to use a movie controller component. If you want to assign a movie to an existing controller, you can use `MCNewAttachedController` (II–823). Use `MCSetMovie` to assign a movie to or remove a movie from a controller. You can use `MCGetIndMovie` (II–812) to retrieve a reference to the movie that is assigned to a controller. When you are done with a controller, call `DisposeMovieController` (I–270) to dispose of it.

Functions

`NewMovieController` (II–1071)

Locates a movie controller component and assigns a movie to that controller.

`MCNewAttachedController` (II–823)

Associates a specified movie with a movie controller.

`MCSetMovie` (II–835)

Associates a movie with a specified movie controller.

`MCGetIndMovie` (II–812)

Lets your application to retrieve the movie reference for a movie that is associated with a movie controller.

`DisposeMovieController` (I–270)

Disposes of a movie controller.

Managing Controller Attributes

Movie controller components provide a number of functions that allow your application to control the display attributes of a movie controller. For example, you can detach the controller from its movie, so that the controller and movie can be managed as separate graphics entities. In addition, movie controller components provide a number of functions that allow you to work with a controller's boundary rectangles and regions.

`MCSetControllerAttached` (II-831) lets you control whether the movie controller is attached to its movie. `MCIsControllerAttached` (II-818) allows you to determine if a controller is attached to its movie. You can use `MCSetControllerPort` (II-833) and `MCGetControllerPort` to work a movie controller's graphics port. `MCSetVisible` (II-837) and `MCGetVisible` (II-815) enable you to control the visibility of the movie controller.

`MCSetControllerBoundsRect` (II-832) and `MCGetControllerBoundsRect` help you work with a movie controller's boundary rectangle. You can use `MCGetControllerBoundsRgn` (II-807) and `MCGetControllerWindowRgn` if the controller is not rectangular. You can position a controller and its movie separately by calling `MCPositionController` (II-824).

Functions

`MCSetControllerAttached` (II-831)

Lets your application control whether a movie controller is attached to its movie or detached from it.

`MCIsControllerAttached` (II-818)

Returns a value that indicates whether a movie controller is attached to its movie.

`MCSetControllerPort` (II-833)

Lets your application set the graphics port for a movie controller.

`MCGetControllerPort` (II-810)

Returns a movie controller's color graphics port.

`MCSetVisible` (II-837)

Lets your application control the visibility of a movie controller.

`MCGetVisible` (II-815)

Returns a value that indicates whether or not a movie controller is visible.

`MCSetControllerBoundsRect` (II–832)

Lets you change the position and size of a movie controller.

`MCGetControllerBoundsRect` (II–806)

Returns a movie controller’s boundary rectangle.

`MCGetControllerBoundsRgn` (II–807)

Returns the actual region occupied by the controller and its movie.

`MCGetWindowRgn` (II–815)

Determines the window region that is actually in use by a controller and its movie.

`MCPositionController` (II–824)

Controls the position of a movie and its controller on the computer display.

`MCSetClip` (II–830)

Lets you set a movie controller’s clipping region.

`MCGetClip` (II–805)

Obtains information describing a movie controller’s clipping regions.

`MCDrawBadge` (II–804)

Displays a controller’s badge.

Handling Movie Events

Movie controller components provide functions that handle movie controller actions. Your application must call these functions whenever an event occurs. If the movie controller component handles the event, your application can loop to wait for the next event. Otherwise, your application must take care of the event as part of its normal event handling.

Movie controller components support an action filter. You can instruct the filter to invoke a function in your application whenever actions occur. This action filter function can then perform specialized processing or refer the action back to the movie controller component. See “Movie Controller Actions” (V–2773).

You can use `MCIsPlayerEvent` (II–819) to handle events for a movie controller. You can obtain information about the current state of the movie controller and its movie by calling `MCGetControllerInfo` (II–808).

Functions

`MCIsPlayerEvent` (II-819)

Handles all events for a movie controller.

`MCGetControllerInfo` (II-808)

Determines the current status of a movie controller and its associated movie, for menu highlighting.

`MCPtInController` (II-826)

Reports whether a point is in the control area of a movie.

See Also

For functions that invoke and filter controller events, see “Movie Controller Actions” (V-2773).

Editing Movies With a Controller

Movie controller components can provide editing capabilities. Your application can use several functions to alter movies that are associated with movie controllers.

Your application uses `MCEnableEditing` (II-805) to enable editing for a specified movie controller. Movie controller components may return an error code indicating that editing is not supported. Use `MCIsEditingEnabled` (II-818) to find out if editing is enabled for a specified controller.

`MCCopy` (II-800), `MCCut` (II-800), `MCPaste` (II-824), `MCClear` (II-798), and `MCUndo` (II-838) support normal editing operations on movies associated with movie controllers. These functions operate on the current movie selection.

Two functions are also provided that facilitate work with Edit menus. You can use `MCSetUpEditMenu` (II-836) to highlight and name the items in the Edit menu for your application, and `MCGetMenuString` (II-814) is provided for you to use with a non-standard Edit menu.

Functions

`MCEnableEditing` (II-805)

Enables and disables editing for a movie controller.

`MCIsEditingEnabled` (II-818)

Determines whether editing is currently enabled for a movie controller.

MCCopy (II-800)

Returns a copy of the current movie selection from the movie associated with a specified controller.

MCCut (II-800)

Returns a copy of the current movie selection from the movie associated with a specified controller and then removes the current movie selection from the source movie.

MCPaste (II-824)

Inserts a specified movie at the current movie time in the movie associated with a specified controller.

MCClear (II-798)

Removes the current movie selection from the movie associated with a specified controller.

MCUndo (II-838)

Lets your application discard the effects of the most recent edit operation.

MCSetUpEditMenu (II-836)

Correctly highlights and names the items in your application's Edit menu.

MCGetMenuString (II-814)

Retrieves the text string for a movie controller menu command.

See Also

For alternatives, see “Editing Movies” (V-2746).

Getting and Setting Movie Controller Time

Movie controller components provide functions that allow your application to work with temporal aspects of movie controllers.

You can use the **MCSetDuration (II-834)** function to set the duration of a movie controller to some arbitrary value. The **MCGetCurrentTime (II-810)** function lets you retrieve the time value represented by the indicator on the movie controller's slider.

Functions

MCSetDuration (II-834)

Lets your application set a controller's duration in the case where a controller does not have a movie associated with it.

`MCGetCurrentTime` (II-810)

Obtains the time value represented by the indicator on the movie controller's slider.

Customizing Event Processing

Movie controller components provide a number of functions that allow your application to customize event processing. If your application does not use `MCIsPlayerEvent` (II-819), you can use these functions to direct movie controller events to the appropriate movie controller component. The component then attempts to handle the event.

Your application obtains the values for many of the function parameters from the appropriate event structure. Each function returns a value that indicates whether it handled the event. If the controller component completely handles the event, the function sets the return value to 1. If the controller component does not handle the event, the function sets the return value to 0, and your application must then handle the event.

Your application can use the functions listed below for customized event processing.

Functions

`MCActivate` (II-795)

Lets a controller respond to activate, deactivate, suspend, and resume events.

`MCClick` (II-799)

Lets a controller respond when the user clicks in a movie controller window.

`MCDraw` (II-803)

Responds to an update event.

`MCIdle` (II-816)

Performs idle processing for a movie controller.

`MCKey` (II-821)

Handles keyboard events for a movie controller.

See Also

For functions that invoke and filter controller events, see “Movie Controller Actions” (V-2773).

Managing Components

Besides the Movie Toolbox, the QuickTime system software contains some 200 or so components. Each component is a piece of QuickTime code that provides a defined set of services to one or more clients. Applications, system extensions, and other components use these services. Multiple components can provide the same type of service, but all components of the same type must support the same basic interface. This allows your application to use the same interface for any given type of component and get the same type of service, yet allows your application to obtain different levels of service.

The Component Manager allows your application to find and utilize various components at run time. It provides access to components and manages them by keeping track of the currently available components and routing requests to the appropriate one. You can also create your own components and use the Component Manager to help manage them.

Component Data Structures

Every component type has its own data structures, by which components of that type communicate with their clients; for further information, see the documentation for each component type. In addition, there is one structure that all components use to describe themselves through the Component Manager.

Your application can use the `ComponentDescription` (IV-2212) structure to find components that provide specific services or meet other selection criteria. It identifies the characteristics of a component, including the type of services offered by the component and its manufacturer.

Data Structure

`ComponentDescription` (IV-2212)

Describes a class of components by their attributes.

Component Functions Used By Applications

The Component Manager provides functions that allow your application to search for components, gain access to and release components, get detailed information about specific components, and get component error information.

When using Component Manager routines, your application must specify the particular component by either a component identifier or a component instance. The Component Manager identifies each component by a component identifier of type `Component`. It identifies each instance of a component by a component instance of type `ComponentInstance`.

You can use `CountComponents` (I-143) to determine the number of components that match a component description. Use `FindNextComponent` (I-360) to find an individual component that matches a description.

You can use `OpenDefaultComponent` (II-1131) to open a component of a specified component type and subtype, or `OpenComponent` (II-1130) to gain access to a specified component. Use `CloseComponent` (I-100) to close an open component.

Your application can get the registration information for any component using `GetComponentInfo` (I-389). You can use `GetComponentIconSuite` (I-387) to get a handle to the component's icon suite, if any. In addition, for components to which your application already has a connection, your application can obtain the component's version number and also determine whether the component supports a particular request by using `GetComponentVersion` (I-395) and `ComponentFunctionImplemented`.

Finally, the Component Manager provides a routine that allows your application to retrieve the last error code that was generated by a component instance. Some component routines return error information as their function result. Other component routines set an error code that your application can retrieve using `GetComponentInstanceError` (I-390).

Functions

`CountComponents` (I-143)

Determines the number of registered components that meet a given set of criteria.

`FindNextComponent` (I–360)

Returns the component identifier for the next registered component that meets the selection criteria specified by an application.

`OpenDefaultComponent` (II–1131)

Gains access to the services provided by a component specified by component type and subtype.

`OpenADefaultComponent` (II–1129)

Gains access to the services provided by a component, specified by component type and subtype, and returns an `OSErr` value.

`OpenComponent` (II–1130)

Gains access to the services provided by a component specified by a component identifier.

`OpenAComponent` (II–1126)

Gains access to the services provided by a component specified by a component identifier and returns an `OSErr` value.

`CloseComponent` (I–100)

Terminates your application's access to the services provided by a component.

`GetComponentInfo` (I–389)

Returns all of the registration information for a component.

`GetComponentIconSuite` (I–387)

Returns a handle to the component's icon suite (if it provides one).

`GetComponentVersion` (I–395)

Returns a component's version number.

`ComponentFunctionImplemented` (I–111)

Determines whether a component supports a specified request.

`GetComponentInstanceError` (I–390)

Returns the last error generated by a specific connection to a component.

Functions Used By Components

The Component Manager provides several routines that components can use to communicate with applications and with one another. To use these routines, a component must first be registered.

Both applications and components can use either `RegisterComponent` (III–1451) or `RegisterComponentResource` to register components, or they can use `RegisterComponentResourceFile` (III–1455) to register all components specified in a given **resource** file. All components stored in a component file must provide a `ComponentResource` (IV–2219) structure.

You can use `UnregisterComponent` (III–1979) to remove a component from the registration list, and `GetComponentListModSeed` (I–391) to determine whether the list of registered components has changed. A component can use `SetDefaultComponent` (III–1581) to change the order in which the list of registered components is searched.

When an application requests service from your component, your component receives a `ComponentParameters` (IV–2216) structure containing the information for that request. Your component can use this structure to access the parameters directly, or it can use certain routines to extract those parameters and pass them to one of its subroutines. You can use `CallComponentFunction` (I–61) to call a component subroutine without providing it access to global data. You can use `CallComponentFunctionWithStorage` (I–61) to call a component subroutine while passing it a handle to global data for that connection.

Use `SetComponentInstanceStorage` (III–1572) to inform the Component Manager of the memory your component is using to maintain global data for a connection, and `GetComponentInstanceStorage` (I–391) to retrieve a handle to that storage. Use `CountComponentInstances` (I–142) to count all the connections that are currently maintained by your component.

In general, your component returns error information (0 or nonzero) in its function result; however, some requests require that your component return other information. In these cases, your component can use `SetComponentInstanceError` (III–1571) to report its latest error state. It can also use this function to report an error at any time during component execution.

Each component is assigned a reference constant, a 4-byte value that it can use in any way. Typically you use the reference constant to store the address of a data structure that is shared by all connections maintained by your component. Use `SetComponentRefcon` (III–1573) to set the value of the reference constant, and use `GetComponentRefcon` (I–394) to retrieve its value.

If you store your component in a component resource and register it, or if the Component Manager automatically registers your component, your component can gain read-only access to its resource file. Use `OpenComponentResFile` (II–1130) to

open your component's resource file, and `CloseComponentResFile` (I-101) to close the resource file before returning to the calling application.

`DelegateComponentCall` (I-249) allows your component to pass a request to a specified component. For example, you might want to create two similar components that provide different levels of service to applications. Rather than completely implementing both components, you could design one to rely on the capabilities of the other. You can also capture and override a component, removing it from the list of available components. Use `CaptureComponent` (I-74) to capture a component and `UncaptureComponent` (III-1979) to restore a previously captured component to the search list. Your component can target a component instance without capturing the component or your component can first capture the component and then target a specific instance of the component. To target a component instance, use `ComponentSetTarget` (I-112).

Functions

`RegisterComponent` (III-1451)

Makes a component that is resident in memory available for use by applications or other clients.

`RegisterComponentResource` (III-1453)

Makes a component that is stored in a file available for use by applications or other clients.

`RegisterComponentResourceFile` (III-1455)

Registers all component **resources** in a given resource file.

`UnregisterComponent` (III-1979)

Removes a component from the Component Manager's registration list.

`GetComponentListModSeed` (I-391)

Determines if the list of registered components has changed.

`GetComponentTypeModSeed` (I-395)

Determines if the specified type of registered component has changed.

`SetDefaultComponent` (III-1581)

Changes the search order for registered components.

`CallComponentFunction` (I-61)

Invokes a specified function of your component with the parameters originally provided by the application that called your component.

CallComponentFunctionWithStorage (I-61)

Invokes a specified function of your component with the parameters originally provided by the application that called your component and provides a handle to the memory associated with the current connection.

CallComponentFunctionWithStorageProcInfo (I-62)

Invokes a specified function of your component with the parameters originally provided by the application that called it; used when your component links with ComponentsInterfacesLib.

SetComponentInstanceStorage (III-1572)

Lets a component pass the Component Manager a handle to the memory that was allocated for the connection to it.

GetComponentInstanceStorage (I-391)

Lets your component retrieve a handle to the memory associated with a component connection.

CountComponentInstances (I-142)

Allows you to determine the number of open connections being managed by a specified component.

SetComponentInstanceError (III-1571)

Lets a component pass error information to the Component Manager other than through its return value.

SetComponentRefcon (III-1573)

Sets the reference constant for a component.

GetComponentRefcon (I-394)

Retrieves the value of the reference constant for your component.

OpenComponentResFile (II-1130)

Allows your component to gain access to its **resource** file.

OpenAComponentResFile (II-1127)

Allows your component to gain access to its **resource** file and returns an OSErr value.

CloseComponentResFile (I-101)

Closes the **resource** file that your component opened previously with the OpenComponentResFile function.

`DelegateComponentCall` (I–249)

Provides an efficient mechanism for passing on requests to a specified component.

`CaptureComponent` (I–74)

Allows your component to capture another component.

`UncaptureComponent` (III–1979)

Uncaptures a previously captured component.

`ComponentSetTarget` (I–112)

Calls a component's target request routine, which handles the `kComponentTargetSelect` request code.

Data Structures

`ComponentParameters` (IV–2216)

Data structure passed by the Component Manager to a component.

`ComponentResource` (IV–2219)

Defines the structure of a component **resource**.

Managing Image Compression

The Image Compression Manager provides image-compression and image-decompression services to applications and other managers. Image compression benefits you by decreasing the amount of storage required for image data, decreasing the time required to exchange image data across networks, and decreasing the time required to read data from disks and CD-ROM volumes.

Image Compression Data Structures

The Image Compression Manager uses several data structures to communicate with your application.

An `ImageDescription` (IV–2285) structure contains information that defines the characteristics of a compressed image or sequence. Data in the image description structure indicates the type of compression that was used, the size of the image when displayed, the resolution at which the image was captured, and so on. One

image description structure may be associated with one or more compressed frames.

A `CodecInfo` (IV-2202) structure contains compressor information. A `CodecNameSpec` (IV-2208) structure contains a compressor's name, and a `CodecNameSpecList` (IV-2208) structure contains a list of such names.

Data Structures

`ImageDescription` (IV-2285)

Defines the characteristics of a compressed image or sequence.

`CodecInfo` (IV-2202)

Describes the capabilities of a compressor.

`CodecNameSpec` (IV-2208)

Defines a compressor name.

`CodecNameSpecList` (IV-2208)

Contains a list of `CodecNameSpec` (IV-2208) structures.

Getting Information About Compressed Data

Several functions enable your application to collect information about compressed images and images that are about to be compressed. Your application may use some of these functions in preparation for compressing or decompressing an image or sequence.

You can use `GetCompressionTime` (I-401) to determine how long it will take for a compressor to compress a specified image. Similarly, you can use `GetMaxCompressionSize` (I-420) to find out how large the compressed image may be after the compression operation. You can use `GetCompressedImageSize` (I-396) to determine the size of a compressed image that does not have a complete image description.

`GetSimilarity` (I-508) allows you to determine how similar two images are. This information is useful when you are performing temporal compression on an image sequence.

Functions

`GetCompressionTime` (I-401)

Determines the estimated amount of time required to compress a given image.

`GetMaxCompressionSize` (I-420)

Determines the maximum size an image will be after compression.

`GetCompressedImageSize` (I-396)

Determines the size, in bytes, of a compressed image.

`GetSimilarity` (I-508)

Compares a compressed image to a picture stored in a pixel map and returns a value indicating the relative similarity of the two images.

Getting Information About Compressor Components

The Image Compression Manager provides functions that let you get information about itself and about the available codecs.

You can use `CodecManagerVersion` (I-103) to retrieve the version number associated with the Image Compression Manager that is installed on a particular computer. You can use `FindCodec` (I-358), `GetCodecInfo` (I-385), and `GetCodecNameList` (I-386) to locate and retrieve information about the compressor components that are available.

Functions

`CodecManagerVersion` (I-103)

Determines the version of the installed Image Compression Manager.

`FindCodec` (I-358)

Determines which of the installed compressors or decompressors has been chosen to field requests made by using one of the special compressor identifiers.

`GetCodecInfo` (I-385)

Returns information about a single compressor component.

`GetCodecNameList` (I-386)

Retrieves a list of installed compressor components or types.

`DisposeCodecNameList` (I-257)

Disposes of the compressor name list structure you obtained by calling `GetCodecNameList` (I-386).

Working With Pixel Maps

The Image Compression Manager provides two levels of functionality for compressing and decompressing single-frame images stored as pixel maps.

If you do not need to assert a lot of control over the compression operation, you can use `CompressImage` (I–113) and `DecompressImage` (I–235) to work with compressed images. If you need more control over the compression parameters, you can use `FCompressImage` (I–344) and `FDecompressImage` (I–355).

You can convert a compressed image from one compression format to another by calling `ConvertImage` (I–131). You can alter the spatial characteristics of a compressed image by calling `TrimImage` (III–1956). You can work with an image's color table using `SetImageDescriptionCTable` (III–1593) and `GetImageDescriptionCTable`.

Functions

`CompressImage` (I–113)

Compresses a single-frame image that is currently stored as a pixel map structure.

`DecompressImage` (I–235)

Decompresses a single-frame image into a pixel map structure.

`FCompressImage` (I–344)

Compresses a single-frame image that is currently stored as a pixel map structure, with added control over the compression process.

`FDecompressImage` (I–355)

Decompresses a single-frame image into a pixel map structure, with added control over the decompression process.

`ConvertImage` (I–131)

Converts the format of a compressed image.

`TrimImage` (III–1956)

Adjusts a compressed image to the boundaries defined by a specified rectangle.

`SetImageDescriptionCTable` (III–1593)

Updates the custom `ColorTable` (IV–2210) structure for an image.

`GetImageDescriptionCTable` (I–417)

Sets the custom color table for an image.

Working With Image Descriptions

The Image Compression Manager provides several functions that let you modify `ImageDescription` (IV–2285) structures.

These functions are listed below.

Functions

`AddImageDescriptionExtension` (I–25)

Adds an extension to an `ImageDescription` (IV–2285) structure.

`GetNextImageDescriptionExtensionType` (I–501)

Retrieves an image description structure extension type.

`CountImageDescriptionExtensionType` (I–143)

Counts the number of extensions of a given type in an `ImageDescriptionHandle`.

`GetImageDescriptionExtension` (I–418)

Returns a new handle with the data from a specified image description extension.

`RemoveImageDescriptionExtension` (III–1457)

Removes a specified extension from an `ImageDescription` (IV–2285) structure.

Working With Pictures and PICT Files

The Image Compression Manager provides two levels of functionality for compressing and decompressing single-frame images stored as pictures and PICT files.

If you do not need to control the compression parameters, use `CompressPicture` (I–116) or `CompressPictureFile` (I–118). If you need more control over the operation, use `FCompressPicture` (I–348) or `FCompressPictureFile` (I–352).

The Image Compression Manager automatically expands compressed pictures when you display them. Use `DrawPictureFile` (I–310) to display the contents of a **picture file**. If you want to alter the spatial characteristics of the image, use `DrawTrimmedPicture` (I–311) or `DrawTrimmedPictureFile`. You can work with an image's control information by calling `GetPictureFileHeader` (I–504).

Functions

CompressPicture (I-116)

Compresses a single-frame image stored as a picture structure and places the result in another picture.

CompressPictureFile (I-118)

Compresses a single-frame image stored as a **picture file** and places the result in another picture file.

FCompressPicture (I-348)

Compresses a single-frame image stored as a picture structure and places the result in another picture, with added control over the compression process.

FCompressPictureFile (I-352)

Compresses a single-frame image stored as a **picture file** and places the result in another picture file, with added control over the compression process.

DrawPictureFile (I-310)

Draws an image from a specified **picture file** in the current graphics port.

DrawTrimmedPicture (I-311)

Draws an image that is stored as a picture into the current graphics port and trims that image to fit a specified region.

DrawTrimmedPictureFile (I-313)

Draws an image that is stored as a **picture file** into the current graphics port and trims that image to fit a specified region.

GetPictureFileHeader (I-504)

Extracts the picture frame and file header from a specified **picture file**.

Making Thumbnail Pictures

The Image Compression manager provides functions that allow your application to create thumbnail pictures from existing images that are stored as pixel maps, pictures, or **picture files**. Thumbnail pictures are useful for creating small, representative images of a source image. You can use thumbnails when you create **previews** for files that contain image data.

Use `MakeThumbnailFromPicture` (II-790), `MakeThumbnailFromPictureFile`, or `MakeThumbnailFromPixMap` (II-792), as appropriate.

Functions

MakeThumbnailFromPicture (II–790)

Creates a thumbnail picture from a specified `Picture` (IV–2331) structure.

MakeThumbnailFromPictureFile (II–791)

Creates a thumbnail picture from a specified **picture file**.

MakeThumbnailFromPixMap (II–792)

Creates a thumbnail picture from a specified `PixMap` (IV–2332) structure.

Working With Sequences

The Image Compression Manager provides a rich set of functions that allow your application to compress and decompress sequences of images. Each image in the sequence is referred to as a frame. Note that the sequence carries no time information; the Movie Toolbox manages all temporal aspects of displaying the sequence. Consequently, your application can focus on the order of images in the sequence.

To process a sequence of frames, your program begins by calling either `CompressSequenceBegin` (I–119), `DecompressSequenceBegin`, or `DecompressSequenceBeginS` (I–239). You then process each frame in the sequence. Use `CompressSequenceFrame` (I–124) to compress a frame; use `DecompressSequenceFrame` (I–242) to decompress a frame. When you are done, close the sequence by calling `CDSequenceEnd` (I–77). You can check on the status of the current operation by calling `CDSequenceBusy` (I–75).

Functions

`CompressSequenceBegin` (I–119)

Signals the beginning of the process of compressing a sequence of frames.

`DecompressSequenceBegin` (I–238)

Obsolete. See `DecompressSequenceBeginS` (I–239).

`DecompressSequenceBeginS` (I–239)

Sends a sample image to a decompressor.

`CompressSequenceFrame` (I–124)

Compresses one of a sequence of frames.

`DecompressSequenceFrame` (I–242)

Obsolete. See `DecompressSequenceFrames` (I–243).

`DecompressSequenceFramesS` (I-243)

Queues a frame for decompression and specifies the size of the compressed data; new applications should use `DecompressSequenceFrameWhen`.

`DecompressSequenceFrameWhen` (I-245)

Queues a frame for decompression and specifies the time at which decompression will begin.

`CDSequenceEnd` (I-77)

Indicates the end of processing for an image sequence.

`CDSequenceBusy` (I-75)

Checks the status of an asynchronous compression or decompression operation.

`CDSequenceFlush` (I-80)

Stops a decompression sequence, aborting processing of any queued frames.

`CDSequenceEquivalentImageDescription` (I-78)

Reports whether two image descriptions are the same.

`CDSequenceNewMemory` (I-84)

Requests codec-allocated memory.

`CDSequenceDisposeMemory` (I-77)

Disposes of memory allocated by the codec.

`CDSequenceInvalidate` (I-82)

Notifies the Image Compression Manager that the destination port for the given image decompression sequence has been invalidated.

`CDSequenceNewDataSource` (I-82)

Creates a new data source.

`CDSequenceDisposeDataSource` (I-76)

Disposes of a data source.

`CDSequenceSetSourceData` (I-86)

Sets data in a new frame to a specific data source.

`CDSequenceChangedSourceData` (I-76)

Notifies the compressor that the image source data has changed.

`SetSequenceProgressProc` (III-1639)

Installs a progress procedure for a sequence.

Changing Sequence-Compression Parameters

The Image Compression Manager provides functions that allow your application to manipulate the parameters that control sequence compression and to get information about memory that the compressor has allocated. You can use these functions during the sequence-compression process. Some of these functions deal with parameter values that cannot be set when starting a sequence.

You can determine the location of the previous image buffer used by the Image Compression Manager by calling `GetCSequencePrevBuffer` (I–406).

You can set a number of compression parameters. Use `SetCSequenceFlushProc` (III–1575) to assign a data-unloading function to the operation. You can set the rate at which the Image Compression Manager inserts **key frames** into the compressed sequence by calling `SetCSequenceKeyFrameRate` (III–1577). You can set the frame against which the compressor compares a frame when performing temporal compression by calling `SetCSequencePrev` (III–1579). Finally, you can control the quality of the compressed image by calling `SetCSequenceQuality` (III–1580).

Functions

`GetCSequencePrevBuffer` (I–406)

Determines the location of the previous image buffer allocated by the compressor.

`GetCSequenceMaxCompressionSize` (I–405)

Determines the maximum size an image will be after compression for a given compression sequence.

`SetCSequenceFlushProc` (III–1575)

Assigns a data-unloading function to a sequence.

`SetCSequenceKeyFrameRate` (III–1577)

Adjusts the **key frame** rate for the current sequence.

`SetCSequencePrev` (III–1579)

Allows the application to set the pixel map and boundary rectangle used by the previous frame in temporal compression.

`SetCSequenceQuality` (III–1580)

Adjusts the spatial or temporal quality for the current sequence.

`SetCSequencePreferredPacketSize` (III–1578)

Sets the preferred packet size for a sequence.

Callback

ICMFlushProc (III–2091)

Writes compressed data to a storage device during a compression operation.

Constraining Compressed Data

The Image Compression Manager provides two functions and a data structure that allow your application to communicate information to compressors that can constrain compressed data to a specific data rate.

A compressor indicates that it can constrain the data rate by setting the `codecInfoDoesRateConstrain` flag in its `CodecInfo` (IV–2202) structure. `SetCSequenceDataRateParams` (III–1574) allows you to specify the parameters in the `DataRateParams` (IV–2231) structure and `GetCSequenceDataRateParams` (I–403) allows you to retrieve these parameters.

Functions

`SetCSequenceDataRateParams` (III–1574)

Communicates information to compressors that can constrain compressed data in a particular sequence to a specific data rate.

`GetCSequenceDataRateParams` (I–403)

Obtains the data rate parameters previously set with `SetCSequenceDataRateParams` (III–1574).

Data Structure

`DataRateParams` (IV–2231)

Communicates information to compressors that can constrain compressed data to a specific data rate.

See Also

See “Image Compression Data Structures” (V–2786).

Changing Sequence-Decompression Parameters

The Image Compression Manager provides functions that enable your application to manipulate the parameters that control sequence decompression and to get information about memory that the decompressor has allocated. Some of these functions deal with parameter values that cannot be set when starting a sequence.

Use `GetDSequenceImageBuffer` (I–409) to determine the location of the image buffer.
Use `GetDSequenceScreenBuffer` (I–410) to determine the location of the screen buffer.

Use `SetDSequenceAccuracy` (III–1583) to control the accuracy of the decompression.
Use `SetDSequenceDataProc` (III–1584) to assign a data-loading function to the operation. Use `SetDSequenceMask` (III–1586) to set the clipping region for the resulting image. You can establish a blend matte for the operation by calling `SetDSequenceMatte` (III–1588). You can alter the spatial characteristics of the resulting image by calling `SetDSequenceMatrix` (III–1587). Your application can establish the size and location of the operation’s source rectangle by calling `SetDSequenceSrcRect` (III–1589). Finally, you can set the transfer mode used by the decompressor when it draws to the screen by calling `SetDSequenceTransferMode` (III–1591).

Functions

`GetDSequenceImageBuffer` (I–409)

Determines the location of the offscreen image buffer allocated by a decompressor.

`GetDSequenceScreenBuffer` (I–410)

Determines the location of the offscreen screen buffer allocated by a decompressor.

`SetDSequenceAccuracy` (III–1583)

Adjusts the decompression accuracy for the current sequence.

`SetDSequenceDataProc` (III–1584)

Assigns a data-loading function to a sequence.

`SetDSequenceMask` (III–1586)

Assigns a clipping region to a sequence.

`SetDSequenceMatte` (III–1588)

Assigns a blend matte to a sequence.

`SetDSequenceMatrix` (III–1587)

Assigns a mapping matrix to a sequence.

`SetDSequenceSrcRect` (III–1589)

Defines the portion of an image to decompress.

`SetDSequenceTransferMode` (III–1591)

Sets the mode used when drawing a decompressed image.

`SetDSequenceTimeCode` (III–1590)

Sets the timecode value for a frame that is about to be decompressed.

`PtInDSequenceData` (II–1147)

Tests to see if a compressed image contains data at a given point.

Callback

`ICMDataProc` (III–2090)

Supplies compressed data during a decompression operation.

Working With the StdPix Function

The `StdPix` (III–1912) function extends the `grafProcs` field of the `CGrafPort` (IV–2168) structure to support compressed data, mattes, and matrices.

To work with the control information associated with a compressed image, you can use `SetCompressedPixMapInfo` (III–1573) and `GetCompressedPixMapInfo`.

Functions

`StdPix` (III–1912)

Extends the `grafProcs` field of the `CGrafPort` (IV–2168) structure to support compressed data, mattes, matrices, and pixel maps, letting you intercept image data in compressed form before it is decompressed and displayed.

`SetCompressedPixMapInfo` (III–1573)

Stores information about a compressed image for `StdPix` (III–1912).

`GetCompressedPixMapInfo` (I–397)

Retrieves information about a compressed image.

Data Structure

`CGrafPort` (IV–2168)

Defines a complete drawing environment for color graphics operations.

Aligning Windows

The Image Compression Manager provides functions that allow your application to position and drag windows to optimal screen positions based on the depth of the screen.

`AlignWindow` (I-47) enables you to transport a specified window to the nearest optimal alignment position. `DragAlignedWindow` (I-306) drags the specified window along an optimal alignment grid. `DragAlignedGrayRgn` (I-304) drags a specified gray region along an optimal alignment grid. `AlignScreenRect` (I-46) aligns a specified rectangle to the strictest screen that the rectangle intersects. The alignment behavior provided by these functions is adequate in the vast majority of situations. However, if you need customized alignment behavior (for example, justification specifications geared to particular video hardware), you can use the `ICMAAlignmentProc` (III-2086) callback.

Functions

`AlignWindow` (I-47)

Moves a specified window to the nearest optimal alignment position.

`DragAlignedWindow` (I-306)

Drags the specified window along an optimal alignment grid.

`DragAlignedGrayRgn` (I-304)

Drags a specified gray region along an optimal alignment grid.

`AlignScreenRect` (I-46)

Aligns a specified rectangle to the strictest screen that the rectangle intersects.

Callback

`ICMAAlignmentProc` (III-2086)

Provides an alignment behavior for windows based on the screen's bit depth.

Working With Graphics Devices and Graphics Worlds

Two Image Compression Manager functions enable you to select graphics devices and create graphics worlds.

You can use `GetBestDeviceRect` (I-382) to select the best available graphics device. `NewImageGWorld` (II-1066) allows you to create a graphics world based on the width, height, depth, and color table of a specified `ImageDescription` (IV-2285) structure.

Functions

`GetBestDeviceRect` (I-382)

Selects the deepest of all available graphics devices, while treating 16-bit and 32-bit screens as having equal depth.

NewImageGWorld (II-1066)
Creates an offscreen graphics world.

Image Compression Progress and Completion Callbacks

The Image Compression Manager defines two callbacks that you may provide to compressor components.

The ICMProgressProc (III-2093) callback allows compressors and decompressors to report that asynchronous operations have completed. The ICMCompletionProc (III-2087) callback provides a mechanism for compressors and decompressors to report their progress toward completing an operation.

Callbacks

ICMProgressProc (III-2093)
Reports on the progress of a compressor or decompressor.

ICMCompletionProc (III-2087)
Called by a compressor component upon completion of an asynchronous operation.

Image Compression Manager Utility Functions

The Image Compression Manager supports two general utility functions.

These functions are listed below.

Functions

ICMShieldSequenceCursor (II-679)
Hides the cursor during decompression operations.

ICMDecompressComplete (II-671)
Signals the completion of a decompression operation.

Using Image-Compression Dialogs

Standard image-compression dialog components provide a consistent user interface for selecting parameters that govern the compression of an image or image sequence and the management of the compression operation. Applications that use these components are freed from many of the details of obtaining and validating image-compression parameters and interacting with the Image Compression Manager to compress an image or sequence.

Getting Default Settings for an Image or a Sequence

The Image Compression Manager provides functions that allow you to derive sensible default compression settings for an image or a sequence. The standard dialog component examines an image you provide and selects appropriate default settings based on the image's characteristics. The component stores those settings for you and uses them with other functions. If you choose to display a dialog box to the user, the component uses these settings as the default dialog box settings.

SCDefaultPictHandleSettings (III–1540) works with pictures,
SCDefaultPictFileSettings (III–1540) works with **picture files**, and
SCDefaultPixMapSettings (III–1541) works with pixel maps.

Functions

SCDefaultPictHandleSettings (III–1540)

Derives default compression settings for a `Picture` (IV–2331) structure that is stored by a handle.

SCDefaultPictFileSettings (III–1540)

Derives default compression settings for a `Picture` (IV–2331) structure that is stored in a file.

SCDefaultPixMapSettings (III–1541)

Derives default compression settings for an image that is stored in a pixel map.

Displaying the Standard Image-Compression Dialog Box

Standard image-compression dialog components provide two functions that allow you to display the standard dialog box to the user and retrieve the compression parameters specified by the user.

Use `SCRequestImageSettings` (III–1555) to retrieve the user’s preferences for compressing a single image; use `SCRequestSequenceSettings` (III–1556) when you are working with an image sequence. You may customize the dialog boxes by specifying a modal-dialog hook function or a custom button. You may use the custom button to invoke an ancillary dialog box that is specific to your application.

Functions

`SCRequestImageSettings` (III–1555)

Displays the standard image dialog box to the user and shows default settings you have established.

`SCRequestSequenceSettings` (III–1556)

Displays the standard sequence dialog box to the user and shows default settings you have established.

Compressing Still Images

The standard dialog component provides three functions you may use to compress a still image. All of these functions use the current compression settings.

`SCCompressImage` (III–1530) works with pixel maps; `SCCompressPicture` (III–1531) compresses a picture that is stored in a handle; and `SCCompressPictureFile` (III–1532) works with pictures stored in files.

Functions

`SCCompressImage` (III–1530)

Compresses an image that is stored in a `PixelFormat` (IV–2332) structure.

`SCCompressPicture` (III–1531)

Compresses a `Picture` (IV–2331) structure that is stored by a handle.

`SCCompressPictureFile` (III–1532)

Compresses a `Picture` (IV–2331) structure that is stored in a file.

Compressing Image Sequences

The standard dialog component provides three functions you may use to compress an image sequence. The standard dialog component manages all of the compression details for you. Your application may have only one sequence-compression operation active on any given connection; naturally, you may have more than one connection active at a time. All of these functions use the current compression settings.

`SCCompressSequenceBegin` (III–1533) allows you to start a sequence-compression operation. Use `SCCompressSequenceFrame` (III–1534) for each image in the sequence. End the sequence by calling `SCCompressSequenceEnd` (III–1534).

Functions

`SCCompressSequenceBegin` (III–1533)

Initiates a sequence-compression operation.

`SCCompressSequenceFrame` (III–1534)

Continues a sequence-compression operation.

`SCCompressSequenceEnd` (III–1534)

Ends a sequence-compression operation.

Working With Image or Sequence Settings

The standard dialog component provides two functions that allow you to work with the current compression settings for an image or a sequence of images.

Use `SCGetInfo` (III–1545) to retrieve settings information. `SCSetInfo` (III–1558) enables you to modify the settings. These functions can work with a number of different types of settings information. When you call either function, you specify the type of data you want to work with. Each of these request types requires different parameter data.

Functions

`SCGetInfo` (III–1545)

Retrieves configuration information from the standard dialog component.

`SCSetInfo` (III–1558)

Modifies the standard dialog component's configuration information.

Specifying a Test Image

The standard image-compression dialog component provided by Apple supports a test image. The component uses this image to display the effect of the user's image-compression settings. In this manner, the user can experiment with different settings and see the results of those settings immediately. The component provides three functions that allow you to specify the test image.

Use `SCSetTestImagePictHandle` (III–1566) if your test image is stored in a handle.

Use `SCSetTestImagePictFile` (III–1564) if your test image is in a **picture file**.

`SCSetTestImagePixMap` (III–1568) sets the test image from a pixel map.

Functions

`SCSetTestImagePictHandle` (III–1566)

Sets the dialog box's test image from a `Picture` (IV–2331) structure that is stored in a handle.

`SCSetTestImagePictFile` (III–1564)

Sets the dialog box's test image from a `Picture` (IV–2331) structure that is stored in a **picture file**.

`SCSetTestImagePixMap` (III–1568)

Sets the dialog box's test image from a `Picture` (IV–2331) structure that is stored in a `PixMap` (IV–2332) structure.

Positioning Dialog Boxes and Rectangles

Standard image-compression dialog components provide functions that allow you to position rectangles and dialog boxes. These functions are most useful in helping you to manage dialog boxes that are related to the standard image-compression dialog.

`SCPositionRect` (III–1554) positions a rectangle; `SCPositionDialog` (III–1553) positions a dialog box. The `SCGetBestDeviceRect` (III–1542) function returns information about the best available display device.

Functions

`SCPositionRect` (III–1554)

Positions a rectangle on the screen.

`SCPositionDialog` (III–1553)

Helps position a dialog box on the screen.

`SCGetBestDeviceRect` (III–1542)

Determines the boundary rectangle that surrounds the display device that supports the largest color or grayscale palette.

Creating a Compression Graphics World

The standard dialog component provides a single utility function that you can use to create a graphics world that is appropriate for the current compression settings.

Use `SCNewGWorld` (III–1552) to create such a graphics environment.

Functions

`SCNewGWorld` (III–1552)

Creates a graphics world based on the current compression settings.

Using the Base Image Decompressor

The base image decompressor is a component that performs tasks for other image decompressor components, including the scheduling of asynchronous decompression operations.

Base Image Decompressor Data Structures

The base image decompressor component communicates with image decompressor components through two data structures.

Your image decompressor component uses the `ImageSubCodecDecompressCapabilities` (IV–2288) structure to report its capabilities to the base image decompressor. The `ImageSubCodecDecompressRecord` (IV–2289) structure contains information needed for decompressing a frame.

Data Structures

`ImageSubCodecDecompressCapabilities` (IV–2288)

Returned by an image decompressor component in response to `ImageCodecInitialize` (II–724).

`ImageSubCodecDecompressRecord` (IV–2289)

Contains information needed for decompressing a frame.

Base Image Decompressor Functions

The base image decompressor makes several calls to your image decompressor component during the decompression process.

It calls your image decompressor component's `ImageCodecInitialize` (II–724) function before making any other all calls to your component, and it calls your image decompressor component's `ImageCodecBeginBand` (II–690) function before drawing a **band** or frame. The base image decompressor calls your `ImageCodecPreflight` (II–732) function before decompressing an image, then calls `ImageCodecDrawBand` (II–697) to decompress a band or frame. The `ImageCodecEndBand` (II–704) function notifies your image decompressor component that decompression of a band has finished or that it was terminated by the Image Compression Manager.

If your component supports asynchronous scheduled decompression, the base image decompressor calls its `ImageCodecQueueStarting` (II–733) function before decompressing the frames in the queue, and `ImageCodecQueueStopping` (II–734) to notify your component that the frames in the queue have been decompressed.

Functions

`ImageCodecInitialize` (II–724)

Called before making any other all calls to your component.

`ImageCodecPreflight` (II–732)

Called before decompressing an image, in response to an `ImageCodecPreDecompress` (II–731) call from the Image Compression Manager.

`ImageCodecBeginBand` (II–690)

Called before drawing a **band** or frame; it allows your image decompressor component to save information about a band before decompressing it.

ImageCodecDrawBand (II–697)

Decompresses a **band** or frame.

ImageCodecEndBand (II–704)

Notifies your image decompressor component that decompression of a **band** has finished or that it was terminated by the Image Compression Manager.

ImageCodecQueueStarting (II–733)

Called by the base image decompressor before decompressing the frames in the queue if your image decompressor component supports asynchronous scheduled decompression.

ImageCodecQueueStopping (II–734)

Notifies your component that the frames in the queue have been decompressed, if your image decompressor component supports asynchronous scheduled decompression.

ImageCodecExtractAndCombineFields (II–705)

Performs field operations on video data.

ImageCodecFlush (II–708)

Empties an image decompressor component's input queue of pending scheduled frames.

ImageCodecSetTimeCode (II–738)

Sets the timecode for the next frame that is to be decompressed.

ImageCodecIsImageDescriptionEquivalent (II–725)

Compares image descriptions.

ImageCodecNewMemory (II–729)

Requests codec-allocated memory.

ImageCodecNewImageBufferMemory (II–727)

Asks a codec to allocate memory for an offscreen buffer of non-RGB pixels.

ImageCodecDisposeMemory (II–696)

Disposes codec-allocated memory.

ImageCodecRequestSettings (II–735)

Displays a dialog box containing codec-specific compression settings.

ImageCodecGetSettings (II–719)

Returns the codec settings chosen by the user.

`ImageCodecSetSettings` (II–737)

Sets the settings of an optional image codec dialog box.

`ImageCodecHitTestData` (II–722)

Notifies your codec when the application calls `PtInDSequenceData` (II–1147).

`ImageCodecGetMaxCompressionSizeWithSources` (II–716)

Notifies your codec when an application calls `GetCSequenceMaxCompressionSize` (I–405).

`ImageCodecSourceChanged` (II–739)

Notifies your codec that one of the data sources has changed when an application calls `CDSequenceSetSourceData` (I–86) or `CDSequenceChangedSourceData` (I–76).

Using Data Codec Components

Data codec components enable you to compress and decompress data using a specified compressor component.

Data Codec Functions

Data compressor components have a component type of `DataCompressorComponentType` and data decompressor components have a component type of `DataDecompressorComponentType`. Each component has a unique component subtype.

With `DataCodecCompress` (I–167) and `DataCodecDecompress` (I–170), you can compress and decompress data using a specified compressor component. You can also compress and decompress **sprites** and 3D media samples. Your software calls the `DataCodecBeginInterruptSafe` (I–166) function before performing a compression or decompression operation during interrupt time. When your software has finished a compression or decompression operation that must be performed during interrupt time, it can release any memory of other **resources** used to make the operation safe by calling the `DataCodecEndInterruptSafe` (I–172) function.

Functions

DataCodecCompress (I-167)

Compresses data using the specified compressor component.

DataCodecDecompress (I-170)

Decompresses data using the specified compressor component.

DataCodecBeginInterruptSafe (I-166)

Called before performing a compression or decompression operation during interrupt time.

DataCodecEndInterruptSafe (I-172)

Releases **resources** used by DataCodecBeginInterruptSafe (I-166).

Sequence Grabbing

Sequence grabber components allow applications to obtain digitized data from sources that are external to a Macintosh computer. For example, you can use a sequence grabber component to record video data from a video digitizer. Your application can then request that the sequence grabber store the captured video data in a QuickTime movie.

Sequence Grabber Data Structures

Sequence grabber components use two data structures to communicate with applications and other components.

The SGCompressInfo (IV-2432) structure defines the characteristics of a buffer that contains a captured image that has been compressed. Callback functions use compression information structures to exchange information about compressed images. The SeqGrabFrameInfo (IV-2430) structure defines a frame for a sequence grabber component and sequence grabber channel components. You need to know about this structure only if you are creating a sequence grabber component.

Data Structures

SGCompressInfo (IV–2432)

Defines the characteristics of a buffer that contains a captured image that has been compressed.

SeqGrabFrameInfo (IV–2430)

Provides information about a frame for a sequence grabber component and its sequence grabber channel components.

Configuring Sequence Grabber Components

Sequence grabber components provide a number of functions that allow you to establish the environment for grabbing or **previewing** digitized data. Before you can start a record or a preview operation, you must initialize the sequence grabber component, establish the channels that will be used, define the display environment for the operation, and determine the optimum screen position for the sequence grabber. In addition, if you are performing a record operation, you must define a destination movie file.

You use `SGInitialize` (III–1752) to initialize a sequence grabber component. Then `SGNewChannel` (III–1753) allows you to create channels for the sequence grabber for an operation. You can use `SGNewChannelFromComponent` (III–1756) to create a new channel using a specified channel component. Use `SGDisposeChannel` (III–1695) to dispose of those channels that you are no longer using. You can use `SGGetIndChannel` (III–1720) to retrieve information about the channels that are currently in use by the sequence grabber.

You can use `SGSetGWorld` (III–1792) and `SGGetGWorld` (III–1719) to establish the display environment for the sequence grabber. These functions affect only those channels that work with data that has visual information.

`SGSetDataOutput` (III–1785) and `SGGetDataOutput` (III–1711) allow you to identify the movie file that is currently assigned to the sequence grabber. You only use these functions when you are performing a record operation.

`SGSetDataProc` (III–1787) allows you to assign a data function to a channel. The sequence grabber calls your data function whenever it writes movie data to the output file. `SGGetAlignmentProc` (III–1698) allows you to determine a sequence grabber's optimum screen position to ensure the best performance and appearance.

Functions

SGInitialize (III–1752)

Initializes the sequence grabber component.

SGNewChannel (III–1753)

Creates a sequence grabber channel and assigns a channel component to the channel.

SGNewChannelFromComponent (III–1756)

Creates a sequence grabber channel and assigns a channel component to the channel.

SGDisposeChannel (III–1695)

Removes a channel from a sequence grabber component.

SGGetIndChannel (III–1720)

Collects information about all of the channel components currently in use by a sequence grabber component.

SGSetGWorld (III–1792)

Establishes the graphics port and device for a sequence grabber component.

SGGetGWorld (III–1719)

Determines the graphics port and device for a sequence grabber component.

SGSetDataOutput (III–1785)

Specifies the movie file and options for a sequence grabber record operation.

SGGetDataOutput (III–1711)

Determines the movie file that is currently assigned to a sequence grabber component and the control flags that would govern a record operation.

SGSetDataProc (III–1787)

Specifies a data function for use by the sequence grabber.

SGSetDataRef (III–1787)

Specifies the destination data reference for a record operation.

SGGetDataRef (III–1716)

Determines the data reference currently assigned to a sequence grabber component and the control flags that would govern a record operation.

SGGetAlignmentProc (III–1698)

Obtains information about the best screen positions for a sequence grabber's video image in terms of appearance and maximum performance.

Controlling Sequence Grabber Components

Sequence grabber components provide a full set of functions that allow your application to control the **preview** or record operation. You can use these functions to start and stop the operation, to pause data collection, and to retrieve a reference to the movie that is created during a record operation.

Use `SGStartPreview` (III–1818) to start a **preview** operation. `SGStartRecord` (III–1819) lets you start a record operation. `SGStop` (III–1819) allows you to stop a sequence grabber component. You can instruct the sequence grabber to pause by calling `SGPause` (III–1771). You can determine whether the sequence grabber is paused by calling `SGGetPause` (III–1730).

You grant processing time to the sequence grabber by calling `SGIdle` (III–1751). Be sure to call this function often during record and preview operations. If your application receives an update event during a record or preview operation, you should call `SGUpdate` (III–1821).

You can prepare the sequence grabber for an upcoming preview or record operation by calling `SGPrepare` (III–1772). This function also allows the sequence grabber to verify that it can support the parameters you have specified. By verifying the parameters you want to use, you can improve the startup of preview and record operations. Use `SGRelease` (III–1773) to release system **resources**.

You can retrieve a reference to the movie created by a record operation by calling `SGGetMovie` (III–1724). You can determine the resource ID value assigned to the last movie resource created by the sequence grabber by calling `SGGetLastMovieResID` (III–1722). You can extract a picture from the video source data by calling `SGGrabPict` (III–1748).

Functions

`SGStartPreview` (III–1818)

Instructs the sequence grabber to begin processing data from its channels.

`SGStartRecord` (III–1819)

Instructs the sequence grabber component to begin collecting data from its channels.

`SGStop` (III–1819)

Stops a **preview** or record operation.

SGPause (III–1771)

Suspends or restarts a sequence grabber record or **preview** operation.

SGGetPause (III–1730)

Determines whether the sequence grabber is paused.

SGIdle (III–1751)

Provides processing time for sequence grabber components.

SGUpdate (III–1821)

Informs your component about update events, to update its display.

SGPrepare (III–1772)

Instructs a sequence grabber to get ready to begin a **preview** or record operation.

SGRelease (III–1773)

Instructs the sequence grabber to release any system **resources** it allocated when you called SGPrepare (III–1772).

SGGetMovie (III–1724)

Returns a reference to the movie that contains the data collected during a record operation.

SGGetLastMovieResID (III–1722)

Retrieves the last **resource** ID used by the sequence grabber component.

SGGrabPict (III–1748)

Lets your application obtain a Picture (IV–2331) structure from a sequence grabber component.

SGGetMode (III–1723)

Determines whether a sequence grabber component is in **preview** mode or record mode.

Working With Sequence Grabber Settings

Sequence grabber components can work with channel components and panel components to collect configuration settings from the user. Five functions allow you to direct the sequence grabber to display its settings dialog box to the user and to work with the configuration of each of the grabber's channels.

Use SGSettingsDialog (III–1808) to instruct the sequence grabber to display its settings dialog box to the user. SGSetSettings (III–1801) and SGGetSettings

(III–1731) allow you to retrieve or set the sequence grabber's configuration. `SGSetChannelSettings` (III–1781) and `SGGetChannelSettings` work with the configuration of an individual channel.

Functions

`SGSettingsDialog` (III–1808)

Causes a sequence grabber to display its settings dialog box to the user.

`SGSetSettings` (III–1801)

Configures a sequence grabber and its channels.

`SGGetSettings` (III–1731)

Retrieves the current settings of all channels used by the sequence grabber.

`SGSetChannelSettings` (III–1781)

Configures a sequence grabber channel.

`SGGetChannelSettings` (III–1707)

Retrieves the current settings of a channel used by the sequence grabber.

Working With Sequence Grabber Characteristics

Sequence grabber components provide a number of functions for working with characteristics that apply to the sequence grabber component itself.

Use `SGSetMaximumRecordTime` (III–1796) to limit the duration of a record operation. You can retrieve this time limit by calling `SGGetMaximumRecordTime` (III–1723).

`SGSetFlags` allows you to set control flags that govern an operation. Use `SGGetFlags` (III–1718) to retrieve those flags. You can obtain information about the progress of a record operation by calling `SGGetStorageSpaceRemaining` (III–1736) and `SGGetTimeRemaining`. You can retrieve a reference to the time base used by a sequence grabber component by calling `SGGetTimeBase` (III–1738).

Functions

`SGSetMaximumRecordTime` (III–1796)

Limits the duration of a record operation

`SGGetMaximumRecordTime` (III–1723)

Determines the time limit you have set for a record operation.

SGSetFlags (III–1789)

Passes control information about the current operation to the sequence grabber component.

SGGetFlags (III–1718)

Retrieves a sequence grabber’s control flags.

SGGetStorageSpaceRemaining (III–1736)

Monitors the amount of space remaining for use during a record operation.

SGGetTimeRemaining (III–1739)

Obtains an estimate of the amount of recording time that remains for the current record operation.

SGGetTimeBase (III–1738)

Retrieves a reference to the time base that is being used by a sequence grabber component.

See Also

The characteristics that govern a sequence grabber operation fall into two main categories: those that apply to the sequence grabber component, and those that apply to an individual channel that has been created for the sequence grabber. This section describes functions that apply to the sequence grabber component. See “Working With Channel Characteristics” (V–2815) for programming information about functions that apply to a single channel.

Working With Sequence Grabber Outputs

A sequence grabber output ties a sequence grabber channel to a specified data reference for the output of captured data. It lets you capture to more than one data reference at a time.

If you want to capture movie data into several different files or data references, you must use the functions listed below to do so. Even if you are using outputs, you must still use SGSetDataOutput (III–1785) or SGSetDataRef (III–1787) to identify where the sequence grabber should create the movie **resource**.

Functions

SGNewOutput (III–1757)

Creates a new sequence grabber output.

SGDisposeOutput (III–1696)

Disposes of an existing sequence grabber output.

SGSetChannelOutput (III-1778)

Assigns an output to a channel.

SGSetOutputFlags (III-1797)

Configures an existing sequence grabber output.

SGGetDataOutputStorageSpaceRemaining (III-1713)

Returns the amount of space remaining in the data reference associated with an output.

SGSetOutputNextOutput (III-1800)

Specifies the order in which sequence grabber outputs should be used.

SGGetOutputNextOutput (III-1729)

Returns the next sequence grabber output for the specified output.

SGSetOutputMaximumOffset (III-1799)

Specifies the maximum offset for data written to a specified sequence grabber output.

SGGetOutputMaximumOffset (III-1728)

Returns the maximum offset for data written to the specified sequence grabber output.

SGGetOutputDataReference (III-1727)

Returns information about the data reference associated with the specified sequence grabber output.

Working With Channel Characteristics

Sequence grabber components use channel components to obtain digitized data from external media. After you create a channel for a sequence grabber component, you must configure that channel before you start a **preview** or record operation. The sequence grabber component provides a number of functions that allow you to configure the characteristics of a channel component.

Use SGSetChannelUsage (III-1782) to specify how a channel is to be used. You can restrict a channel to use during record or **preview** operations. In addition, this function allows you to specify whether a channel plays during a record operation. SGGetChannelUsage (III-1709) enables you to determine a channel's usage. SGGetChannelInfo (III-1702) allows you to determine whether a channel has a visual or an audio representation.

`SGSetChannelPlayFlags` (III–1779) allows you to influence the speed and quality with which the sequence grabber displays captured data. `SGGetChannelPlayFlags` (III–1705) lets you determine these flag settings. `SGSetChannelMaxFrames` (III–1777) establishes a limit on the number of frames that the sequence grabber will capture from a channel. `SGGetChannelMaxFrames` (III–1704) allows you to determine that limit.

`SGSetChannelBounds` (III–1774) allows you to set the display boundary rectangle for a channel. Use `SGGetChannelBounds` (III–1700) to determine a channel’s boundary rectangle. `SGSetChannelVolume` (III–1783) allows you to control a channel’s sound volume. Use `SGGetChannelVolume` (III–1710) to determine a channel’s volume.

`SGSetChannelRefCon` (III–1781) allows you to set the value of a reference constant that is passed to your callback functions. Use `SGGetChannelSampleDescription` (III–1706) to retrieve a channel’s sample description. `SGSetChannelTimeScale` (III–1708) allows you to obtain the channel’s time scale. You can modify or retrieve the channel’s clipping region by calling `SGSetChannelClip` (III–1775) or `SGGetChannelClip` (III–1700), respectively. You can work with a channel’s transformation matrix by calling `SGSetChannelMatrix` (III–1776) and `SGGetChannelMatrix`.

Functions

`SGSetChannelUsage` (III–1782)

Specifies how a channel is to be used by the sequence grabber component.

`SGGetChannelUsage` (III–1709)

Determines how the sequence grabber component is using a channel.

`SGGetChannelInfo` (III–1702)

Determines how a channel’s data is represented to the user: as visual data or audio data, or both.

`SGSetChannelPlayFlags` (III–1779)

Adjusts the speed and quality with which the sequence grabber displays data from a channel.

`SGGetChannelPlayFlags` (III–1705)

Retrieves the playback control flags that you set with `SGSetChannelPlayFlags` (III–1779).

`SGSetChannelMaxFrames` (III–1777)

Limits the number of frames that the sequence grabber will capture from a specified channel.

`SGGetChannelMaxFrames` (III–1704)

Determines the number of frames left to be captured from a specified channel.

`SGSetChannelBounds` (III–1774)

Specifies a channel's display boundary rectangle.

`SGGetChannelBounds` (III–1700)

Determines a channel's display boundary rectangle.

`SGSetChannelVolume` (III–1783)

Sets a channel's sound volume.

`SGGetChannelVolume` (III–1710)

Determines a channel's sound volume setting.

`SGSetChannelRefCon` (III–1781)

Sets the value of a reference constant that is passed to your callback functions for channel components.

`SGGetChannelSampleDescription` (III–1706)

Retrieves a channel's sample description structure.

`SGGetChannelTimeScale` (III–1708)

Lets the sequence grabber retrieve a channel's time scale.

`SGSetChannelClip` (III–1775)

Sets a channel's clipping region.

`SGGetChannelClip` (III–1700)

Retrieves a channel's clipping region.

`SGSetChannelMatrix` (III–1776)

Sets a channel's display transformation matrix.

`SGGetChannelMatrix` (III–1703)

Retrieves a channel's display transformation matrix.

See Also

The characteristics that govern a sequence grabber operation fall into two main categories: those that apply to the sequence grabber component, and those that apply to an individual channel that has been created for the sequence grabber. This section describes functions that apply to individual channels. See “Working With Sequence Grabber Characteristics” (V–2813) for programming information about functions that apply to the sequence grabber component itself.

Besides the generic channel functions listed here, sequence grabber components

provide functions that are specific to the channel type. See “Working With Video Channels” (V–2819) for information about the sequence grabber configuration functions that work only with video channels. See “Working With Sound Channels” (V–2821) for information about the sequence grabber configuration functions that work only with sound channels.

Working With Channel Devices

Sequence grabbers provide a number of functions that allow you to determine the device that is attached to a given sequence grabber channel. These devices allow the channel component to control the digitizing equipment. For example, video channels use video digitizer components, and sound channels use sound input drivers. Your application can use these routines to present a list of available devices to the user, allowing the user to select a specific device for each channel.

You may use `SGGetChannelDeviceList` (III–1701) to retrieve a list of devices that may be used with a specified channel. You dispose of this device list by calling `SGDisposeDeviceList` (III–1696). You can place one or more device names into a menu by calling `SGAppendDeviceListToMenu` (III–1685). You can use `SGSetChannelDevice` (III–1776) to assign a device to a channel.

Some of these functions use a `SGDeviceListRecord` (IV–2433) structure to pass information about one or more channel devices.

Functions

`SGGetChannelDeviceList` (III–1701)

Retrieves a list of the devices that are valid for a specified channel.

`SGDisposeDeviceList` (III–1696)

Disposes of a device list.

`SGAppendDeviceListToMenu` (III–1685)

Places a list of device names into a specified menu.

`SGSetChannelDevice` (III–1776)

Assigns a device to a channel.

Data Structure

`SGDeviceListRecord` (IV–2433)

Passes information about one or more channel devices.

Working With Video Channels

Sequence grabber components provide a number of functions that allow you to configure the grabber's video channels. You can use these functions only with video channels. You can determine whether a channel has a visual representation by calling `SGGetChannelInfo` (III–1702).

`SGGetSrcVideoBounds` (III–1735) allows you to determine the coordinates of the source video boundary rectangle. This rectangle defines the size of the source video image being captured by the video channel. You can use `SGSetVideoRect` (III–1816) to specify a part of the source video boundary rectangle to be captured by the channel. `SGGetVideoRect` (III–1746) allows you to determine the active source video rectangle.

Typically, the sequence grabber component uses the Image Compression Manager to compress the video data it captures. You can control many aspects of this image-compression process. Use `SGSetVideoCompressorType` (III–1814) to specify the type of image compressor to use. You can determine the type of image compressor currently in use by calling `SGGetVideoCompressorType` (III–1744). You can specify a particular image compressor and set many image-compression parameters by calling `SGSetVideoCompressor` (III–1812). You can determine which image compressor is being used and its parameter settings by calling `SGGetVideoCompressor` (III–1742).

Sequence grabber components provide functions that allow you to work with a channel's video digitizer component. You can use `SGGetVideoDigitizerComponent` (III–1745) to determine which video digitizer component is supplying data to a specified channel component. You can set a channel's video digitizer by calling `SGSetVideoDigitizerComponent` (III–1815). If you change any video digitizer settings by calling the video digitizer component directly, you should inform the sequence grabber component by calling `SGVideoDigitizerChanged` (III–1822).

Some video source data may contain unacceptable levels of visual noise or artifacts. One technique for removing this noise is to capture the image and then reduce it in size. During the size reduction process, the noise can be filtered out. `SGSetCompressBuffer` (III–1784) sets a filter buffer for a video channel. `SGGetCompressBuffer` (III–1711) returns information about your filter buffer.

You can work with a video channel's frame rate by calling `SGSetFrameRate` (III–1792) and `SGGetFrameRate` (III–1719). You can control whether a channel uses an offscreen buffer by calling `SGSetUseScreenBuffer` (III–1811) and `SGGetUseScreenBuffer`.

Functions

SGGetSrcVideoBounds (III–1735)

Determines the size of the source video boundary rectangle.

SGSetVideoRect (III–1816)

Specifies a part of the source video image that is to be captured by a sequence grabber component.

SGGetVideoRect (III–1746)

Determines the portion of the source video image that is to be captured.

SGSetVideoCompressorType (III–1814)

Specifies the type of image compression to be applied to captured video images.

SGGetVideoCompressorType (III–1744)

Determines the type of image compression that is being applied to a channel's video data.

SGSetVideoCompressor (III–1812)

Specifies many of the parameters that control image compression of the video data captured by a video channel.

SGGetVideoCompressor (III–1742)

Determines a channel's current image compression parameters.

SGGetVideoDigitizerComponent (III–1745)

Determines the video digitizer component that is providing source video to a video channel component.

SGSetVideoDigitizerComponent (III–1815)

Assigns a video digitizer component to a video channel.

SGVideoDigitizerChanged (III–1822)

Notifies the sequence grabber component whenever you change the configuration of a video channel's video digitizer.

SGSetCompressBuffer (III–1784)

Allows the sequence grabber component to direct your component to create a filter buffer for your video channel.

SGGetCompressBuffer (III–1711)

Returns information about the filter buffer established for a video channel.

SGSetFrameRate (III–1792)

Specifies a video channel's frame rate for recording.

`SGGetFrameRate` (III–1719)

Retrieves a video channel's frame rate for recording.

`SGSetUseScreenBuffer` (III–1811)

Controls whether a video channel uses an offscreen buffer.

`SGGetUseScreenBuffer` (III–1740)

Determines whether a video channel is allowed to use an offscreen buffer.

Working With Video Fields

Three functions let you work with compressed fields of video data.

The `ImageFieldSequenceBegin` (II–746) function initiates an image field sequence operation; the `ImageFieldSequenceExtractCombine` (II–747) function performs the desired operations; and the `ImageFieldSequenceEnd` (II–747) function terminates the operation.

Functions

`ImageFieldSequenceBegin` (II–746)

Initiates an image field sequence operation and specifies the input and output data format.

`ImageFieldSequenceExtractCombine` (II–747)

Performs field operations on video data.

`ImageFieldSequenceEnd` (II–747)

Ends an image field sequence operation.

Working With Sound Channels

Sequence grabber components provide a number of functions that allow you to configure the grabber's sound channels. You can use these functions only with sound channels. You can determine whether a channel has a sound representation by calling `SGGetChannelInfo` (III–1702).

Use `SGSetSoundInputDriver` (III–1802) to specify a channel's sound input device. You can determine a channel's sound input device by calling `SGGetSoundInputDriver` (III–1732). If you change any attributes of the sound input device, you should notify the sequence grabber component by calling `SGSoundInputDriverChanged` (III–1817). By default, the sequence grabber component

uses the sound driver's best settings. You can control the amount of sound data the sequence grabber works with at one time by calling `SGSetSoundRecordChunkSize` (III–1805). You can determine this value by calling `SGGetSoundRecordChunkSize` (III–1734).

You can control the rate at which the sound channel samples the input data by calling `SGSetSoundInputRate` (III–1804). You can determine the sample rate by calling `SGGetSoundInputRate` (III–1734). You can control other sound input parameters by using `SGSetSoundInputParameters` (III–1803) and `SGGetSoundInputParameters`.

Functions

`SGSetSoundInputDriver` (III–1802)

Assigns a sound input device to a sound channel.

`SGGetSoundInputDriver` (III–1732)

Determines the sound input device currently in use by a sound channel component.

`SGSoundInputDriverChanged` (III–1817)

Notifies the sequence grabber component whenever you change the configuration of a sound channel's sound input device.

`SGSetSoundRecordChunkSize` (III–1805)

Controls the amount of sound data in each group of sound samples during a record operation.

`SGGetSoundRecordChunkSize` (III–1734)

Determines the amount of sound data the sequence grabber component works with at a time.

`SGSetSoundInputRate` (III–1804)

Sets the rate at which the sound channel obtains its sound data.

`SGGetSoundInputRate` (III–1734)

Determines the rate at which the sound channel is collecting sound data.

`SGSetSoundInputParameters` (III–1803)

Sets various parameters that relate to sound recording.

`SGGetSoundInputParameters` (III–1733)

Retrieves various parameters that relate to sound recording.

Video Channel Callback Functions

Sequence grabber components allow you to define a number of callback functions in your application. The sequence grabber calls your functions at specific points in the process of collecting, compressing, and displaying the source video data. By defining callback functions, you can control the process more precisely or customize the operation of the sequence grabber component.

`SGSetVideoBottlenecks` (III–1811) lets you assign callback functions to a video channel. You can use `SGGetVideoBottlenecks` (III–1741) to determine the callback functions that have been assigned to a video channel. The `VideoBottles` (IV–2501) structure identifies the callback functions to be assigned to the channel.

Your `SGGrabBottleProc` (III–2142) callback is used by the sequence grabber component to begin the capture of a frame of video data. Your `SGGrabCompleteBottleProc` (III–2143) callback allows the sequence grabber component to determine whether the current frame-capture operation is complete.

Your `SGDisplayBottleProc` (III–2140) callback enables the sequence grabber component to move a captured video image in an offscreen buffer into the destination buffer for the video channel. The sequence grabber component uses your `SGCompressBottleProc` (III–2137) callback to commence the compression of a captured video image. Your `SGCompressCompleteBottleProc` (III–2138) callback helps the sequence grabber component to find out if the current frame-compression operation is finished. Your `SGAddFrameBottleProc` (III–2136) callback lets the sequence grabber component add a frame to a movie.

The sequence grabber component uses your `SGTransferFrameBottleProc` (III–2145) callback to move a video frame from the capture buffer into the channel's filter buffer. You may provide two functions for use with compressed-source devices. Your `SGGrabCompressCompleteBottleProc` (III–2143) callback determines when the current capture and compress operation is complete. Your `SGDisplayCompressBottleProc` (III–2141) callback decompresses and displays a frame.

Functions

`SGSetVideoBottlenecks` (III–1811)

Assigns callback functions to a video channel.

`SGGetVideoBottlenecks` (III–1741)

Determines the callback functions that have been assigned to a video channel.

Data Structures

VideoBottles (IV–2501)

Identifies the callback functions to be assigned to a sequence grabber channel.

Using Sequence Grabber Channels

Sequence grabber components use sequence grabber channel components to obtain data from audio- or video-digitizing equipment. These components isolate the sequence grabber component from the details of working with the various types of data that can be collected. The functionality provided by a sequence grabber component depends upon the services provided by sequence grabber channel components. The channel components, in turn, may use other components to interact with the digitizing equipment.

Configuring Sequence Grabber Channel Components

Sequence grabber components use two functions to establish the environment for grabbing or **previewing** digitized data.

The sequence grabber component uses `SGInitChannel` (III–1751) to initialize your channel prior to a record or **preview** operation. `SGSetGWorld` (III–1792) allows the sequence grabber component to assign a graphics world to your component.

Functions

`SGInitChannel` (III–1751)

Initializes a channel component.

`SGSetGWorld` (III–1792)

Establishes the graphics port and device for a sequence grabber component.

Controlling Sequence Grabber Channel Components

Sequence grabber channel components must provide a full set of functions that allow the sequence grabber component to control the **preview** or record operation.

The sequence grabber component can use these functions to start and stop the operation, to pause data collection, and to write captured data to a movie.

The sequence grabber component uses `SGStartPreview` (III–1818) to start a **preview** operation. `SGStartRecord` (III–1819) starts a record operation. `SGStop` (III–1819) stops your channel component after a preview or record operation. The sequence grabber component grants processing time to your channel component by calling `SGIdle` (III–1751). The sequence grabber notifies you of update events by calling `SGUpdate` (III–1821). The sequence grabber pauses the current operation by calling `SGPause` (III–1771).

The sequence grabber component calls `SGWriteSamples` (III–1825) to write captured data to a movie file after a record operation. The sequence grabber component prepares your channel component for an upcoming preview or record operation by calling `SGPrepare` (III–1772). This function also allows the sequence grabber component to verify that your component can support the parameters an application has specified. `SGRelease` (III–1773) releases system **resources** allocated.

Functions

`SGStartPreview` (III–1818)

Instructs the sequence grabber to begin processing data from its channels.

`SGStartRecord` (III–1819)

Instructs the sequence grabber component to begin collecting data from its channels.

`SGStop` (III–1819)

Stops a **preview** or record operation.

`SGIdle` (III–1751)

Provides processing time for sequence grabber components.

`SGUpdate` (III–1821)

Informs your component about update events, to update its display.

`SGPause` (III–1771)

Suspends or restarts a sequence grabber record or **preview** operation.

`SGWriteSamples` (III–1825)

Called by a sequence grabber component when it is ready to add recorded data to a movie.

SGPrepare (III–1772)

Instructs a sequence grabber to get ready to begin a **preview** or record operation.

SGRelease (III–1773)

Instructs the sequence grabber to release any system **resources** it allocated when you called SGPrepare (III–1772).

See Also

See “Controlling Sequence Grabber Components” (V–2811).

Configuration Functions for All Channel Components

A sequence grabber component must provide a number of functions that allow the sequence grabber to configure the characteristics of your channel. Several of these functions work on any channel component.

SGSetChannelUsage (III–1782) specifies how your channel is to be used. The sequence grabber component can restrict a channel to use during record or **preview** operations. In addition, this function allows the sequence grabber component to specify whether your channel plays during a record operation. SGGetChannelUsage (III–1709) allows the sequence grabber component to determine a channel’s usage. SGGetChannelInfo (III–1702) allows the sequence grabber component to determine some of the characteristics of your channel. For example, this function returns information indicating whether your channel has a visual or an audio representation.

SGSetChannelPlayFlags (III–1779) lets the sequence grabber component influence the speed and quality with which your channel plays captured data.

SGGetChannelPlayFlags (III–1705) allows the sequence grabber component to determine these flag settings. SGSetChannelMaxFrames (III–1777) establishes a limit on the number of frames that your channel component will capture from a channel. SGGetChannelMaxFrames (III–1704) enables the sequence grabber component to determine that limit.

SGSetChannelRefCon (III–1781) allows the sequence grabber component to set the value of a reference constant that your component passes to its callback functions. SGGetDataRate (III–1715) allows the sequence grabber component to determine how many bytes of captured data your channel is collecting each second. SGGetChannelSampleDescription (III–1706) allows the sequence grabber to retrieve your channel’s sample description.

`SGGetChannelTimeScale` (III–1708) allows it to obtain your channel’s time scale. The sequence grabber can modify or retrieve your channel’s clipping region by calling `SGSetChannelClip` (III–1775) or `SGGetChannelClip` (III–1700), respectively. The sequence grabber can work with your channel’s transformation matrix by calling `SGSetChannelMatrix` (III–1776) and `SGGetChannelMatrix`.

Functions

`SGSetChannelUsage` (III–1782)

Specifies how a channel is to be used by the sequence grabber component.

`SGGetChannelUsage` (III–1709)

Determines how the sequence grabber component is using a channel.

`SGGetChannelInfo` (III–1702)

Determines how a channel’s data is represented to the user: as visual data or audio data, or both.

`SGSetChannelPlayFlags` (III–1779)

Adjusts the speed and quality with which the sequence grabber displays data from a channel.

`SGGetChannelPlayFlags` (III–1705)

Retrieves the playback control flags that you set with `SGSetChannelPlayFlags` (III–1779).

`SGSetChannelMaxFrames` (III–1777)

Limits the number of frames that the sequence grabber will capture from a specified channel.

`SGGetChannelMaxFrames` (III–1704)

Determines the number of frames left to be captured from a specified channel.

`SGSetChannelRefCon` (III–1781)

Sets the value of a reference constant that is passed to your callback functions for channel components.

`SGGetDataRate` (III–1715)

Determines for a sequence grabber how much recording time is left.

`SGGetChannelSampleDescription` (III–1706)

Retrieves a channel’s sample description structure.

`SGGetChannelTimeScale` (III–1708)

Lets the sequence grabber retrieve a channel’s time scale.

SGSetChannelClip (III–1775)

Sets a channel's clipping region.

SGGetChannelClip (III–1700)

Retrieves a channel's clipping region.

SGSetChannelMatrix (III–1776)

Sets a channel's display transformation matrix.

SGGetChannelMatrix (III–1703)

Retrieves a channel's display transformation matrix.

Managing Channel Devices

Sequence grabbers provide two functions that allow applications to determine the devices that can be, or the device that is, attached to a given sequence grabber channel. These devices, in turn, allow the channel component to control the digitizing equipment.

The sequence grabber may use SGGetChannelDeviceList (III–1701) to retrieve a list of devices that may be used by your channel. The sequence grabber can use SGSetChannelDevice (III–1776) to assign a device to your channel. This function uses a SGDeviceListRecord (IV–2433) structure to pass information about one or more channel devices.

Functions

SGGetChannelDeviceList (III–1701)

Retrieves a list of the devices that are valid for a specified channel.

SGSetChannelDevice (III–1776)

Assigns a device to a channel.

Data Structure

SGDeviceListRecord (IV–2433)

Passes information about one or more channel devices.

Configuration Functions for Video Channel Components

Video channel components provide a number of functions that allow the sequence grabber to configure the channel's video characteristics. The sequence grabber

component uses these video channel configuration functions only with video channels.

`SGSetChannelBounds` (III–1774) allows the sequence grabber to set the display boundary rectangle for a video channel. `SGGetChannelBounds` (III–1700) determines a channel's boundary rectangle. The sequence grabber component uses `SGGetSrcVideoBounds` (III–1735) to determine the coordinates of the source video boundary rectangle. This rectangle defines the size of the source video image being captured by a video channel. `SGSetVideoRect` (III–1816) specifies a part of the source video boundary rectangle to be captured by the channel. `SGGetVideoRect` (III–1746) retrieves this active source video rectangle.

Typically, video channel components use the Image Compression Manager to compress the video data they capture. The sequence grabber component can control many aspects of this image-compression process. `SGSetVideoCompressorType` (III–1814) specifies the type of image compressor to use. The sequence grabber can determine the type of image compressor currently in use by calling `SGGetVideoCompressorType` (III–1744). The sequence grabber component can specify a particular image compressor and set many image-compression parameters by calling `SGSetVideoCompressor` (III–1812). The sequence grabber component can determine which image compressor is being used and its parameter settings by calling `SGGetVideoCompressor` (III–1742).

Sequence grabber components provide functions that allow an application to work with a channel's video digitizer component. Video channel components, in turn, must provide support for these functions. The sequence grabber component uses `SGGetVideoDigitizerComponent` (III–1745) to determine which video digitizer component is supplying data to your video channel component. The sequence grabber component sets a channel's video digitizer component by calling `SGSetVideoDigitizerComponent` (III–1815). If an application changes any video digitizer settings by calling the video digitizer component directly, the sequence grabber component informs your video channel component by calling `SGVideoDigitizerChanged` (III–1822).

Some video source data may contain unacceptable levels of visual noise or artifacts. One technique for removing this noise is to capture the image and then reduce it in size. During the size reduction process, the noise can be filtered out. Some video channel components may provide functions that allow the sequence grabber component to filter the input video data. `SGSetCompressBuffer` (III–1784) sets a filter buffer for a video channel. `SGGetCompressBuffer` (III–1711) returns information about your filter buffer.

The sequence grabber can work with a video channel's frame rate by calling `SGSetFrameRate` (III–1792) and `SGGetFrameRate` (III–1719). The sequence grabber can control whether your channel uses an offscreen buffer by calling `SGSetUseScreenBuffer` (III–1811) and `SGGetUseScreenBuffer`. `SGAlignChannelRect` (III–1684) allows the sequence grabber to determine a channel's optimum screen position.

Functions

`SGSetChannelBounds` (III–1774)

Specifies a channel's display boundary rectangle.

`SGGetChannelBounds` (III–1700)

Determines a channel's display boundary rectangle.

`SGGetSrcVideoBounds` (III–1735)

Determines the size of the source video boundary rectangle.

`SGSetVideoRect` (III–1816)

Specifies a part of the source video image that is to be captured by a sequence grabber component.

`SGGetVideoRect` (III–1746)

Determines the portion of the source video image that is to be captured.

`SGSetVideoCompressorType` (III–1814)

Specifies the type of image compression to be applied to captured video images.

`SGGetVideoCompressorType` (III–1744)

Determines the type of image compression that is being applied to a channel's video data.

`SGSetVideoCompressor` (III–1812)

Specifies many of the parameters that control image compression of the video data captured by a video channel.

`SGGetVideoCompressor` (III–1742)

Determines a channel's current image compression parameters.

`SGGetVideoDigitizerComponent` (III–1745)

Determines the video digitizer component that is providing source video to a video channel component.

`SGSetVideoDigitizerComponent` (III–1815)

Assigns a video digitizer component to a video channel.

`SGVideoDigitizerChanged` (III–1822)

Notifies the sequence grabber component whenever you change the configuration of a video channel's video digitizer.

`SGSetCompressBuffer` (III–1784)

Allows the sequence grabber component to direct your component to create a filter buffer for your video channel.

`SGGetCompressBuffer` (III–1711)

Returns information about the filter buffer established for a video channel.

`SGSetFrameRate` (III–1792)

Specifies a video channel's frame rate for recording.

`SGGetFrameRate` (III–1719)

Retrieves a video channel's frame rate for recording.

`SGSetUseScreenBuffer` (III–1811)

Controls whether a video channel uses an offscreen buffer.

`SGGetUseScreenBuffer` (III–1740)

Determines whether a video channel is allowed to use an offscreen buffer.

`SGAlignChannelRect` (III–1684)

Determines whether or not a channel prefers to draw at a particular screen location.

Configuration Functions for Sound Channel Components

Sound channel components provide a number of functions that allow sequence grabber components to configure the component's sound channel. The sequence grabber component uses these functions only with sound channels.

`SGSetChannelVolume` (III–1783) allows the sequence grabber component to control a channel's sound volume. The sequence grabber component uses

`SGGetChannelVolume` (III–1710) to determine a channel's volume.

`SGSetSoundInputDriver` (III–1802) specifies a channel's sound input device. The sequence grabber component can determine a channel's sound input device by calling `SGGetSoundInputDriver` (III–1732). If an application changes any attributes of the sound input device, the sequence grabber component notifies your sound

component by calling `SGSoundInputDriverChanged` (III–1817).

The sequence grabber component can control the amount of sound data your channel works with at one time by calling `SGSetSoundRecordChunkSize` (III–1805). The sequence grabber component can determine this value by calling `SGGetSoundRecordChunkSize` (III–1734). The sequence grabber component controls the rate at which your sound channel samples the input data by calling `SGSetSoundInputRate` (III–1804). The sequence grabber component can determine the sample rate by calling `SGGetSoundInputRate` (III–1734). The sequence grabber can control other sound input parameters by using `SGSetSoundInputParameters` (III–1803) and `SGGetSoundInputParameters`.

Functions

`SGSetChannelVolume` (III–1783)

Sets a channel's sound volume.

`SGGetChannelVolume` (III–1710)

Determines a channel's sound volume setting.

`SGSetSoundInputDriver` (III–1802)

Assigns a sound input device to a sound channel.

`SGGetSoundInputDriver` (III–1732)

Determines the sound input device currently in use by a sound channel component.

`SGSoundInputDriverChanged` (III–1817)

Notifies the sequence grabber component whenever you change the configuration of a sound channel's sound input device.

`SGSetSoundRecordChunkSize` (III–1805)

Controls the amount of sound data in each group of sound samples during a record operation.

`SGGetSoundRecordChunkSize` (III–1734)

Determines the amount of sound data the sequence grabber component works with at a time.

`SGSetSoundInputRate` (III–1804)

Sets the rate at which the sound channel obtains its sound data.

`SGGetSoundInputRate` (III–1734)

Determines the rate at which the sound channel is collecting sound data.

`SGSetSoundInputParameters` (III–1803)

Sets various parameters that relate to sound recording.

`SGGetSoundInputParameters` (III–1733)

Retrieves various parameters that relate to sound recording.

Utility Functions for Sequence Grabber Channel Components

Sequence grabber components provide several utility functions that your channel component can use.

`SGAddMovieData` (III–1682) lets you add data and sample references to a movie. Alternatively, you can use `SGWriteMovieData` (III–1824) to add data to a movie, and `SGAddFrameReference` (III–1681) and `SGGetNextFrameReference` to keep track of sample references prior to creating a QuickTime movie from recorded data. These functions take a pointer to a `SeqGrabFrameInfo` (IV–2430) structure as a parameter.

`SGSortDeviceList` allows you to sort entries in the device list. `SGChangedSource` (III–1686) allows you to tell the sequence grabber that you have changed your source device.

Functions

`SGAddMovieData` (III–1682)

Lets a channel component add data to a movie.

`SGWriteMovieData` (III–1824)

Lets a channel component add data to a movie.

`SGAddFrameReference` (III–1681)

Stores sample references for a channel component.

`SGGetNextFrameReference` (III–1726)

Lets a channel component retrieve the sample references that were stored by calling `SGAddMovieData` (III–1682) or `SGAddFrameReference` (III–1681).

`SGSortDeviceList` (III–1817)

Sorts a device list alphabetically.

`SGChangedSource` (III–1686)

Informs the sequence grabber that a component is now using a different device.

`SGAddExtendedFrameReference` (III–1677)

Stores extended sample references for a channel component.

SGAddExtendedMovieData (III–1678)

Adds data to a movie without writing data to a movie file.

SGAddOutputDataRefToMedia (III–1683)

Manages capture sessions that involve multiple data references.

SGChannelGetDataSourceName (III–1687)

Returns the data source name for a track.

SGChannelGetRequestedDataRate (III–1688)

Returns the current maximum data rate requested for a channel.

SGChannelSetDataSourceName (III–1690)

Sets the data source name for a track.

SGChannelSetRequestedDataRate (III–1691)

Specifies the maximum requested data rate for a channel.

SGGetAdditionalSoundRates (III–1697)

Returns the additional sound sample rates added to a specified sequence grabber sound channel.

SGGetNextExtendedFrameReference (III–1725)

Allows a channel component to retrieve the sample references stored previously by **SGAddExtendedMovieData (III–1678)** or **SGAddExtendedFrameReference (III–1677)**.

SGGetPreferredPacketSize (III–1730)

Returns the preferred packet size for the sequence grabber component.

SGGetUserVideoCompressorList (III–1740)

Returns the video compression formats to be displayed by the specified sequence grabber video channel.

SGSetAdditionalSoundRates (III–1774)

Specifies a list of sound sample rates to be included in the sequence grabber's sound settings dialog box.

SGSetPreferredPacketSize (III–1801)

Sets the preferred packet size for the sequence grabber channel component.

SGSetUserVideoCompressorList (III–1810)

Specifies the list of video compression formats to be included in the sequence grabber's video settings dialog box.

SGWriteExtendedMovieData (III–1822)

Allows your channel component to add data to a movie.

Data Structure

SeqGrabFrameInfo (IV–2430)

Provides information about a frame for a sequence grabber component and its sequence grabber channel components.

Using Sequence Grabber Panels

Sequence grabber components create a settings dialog box that includes items that are managed by sequence grabber panel components and sequence grabber channel components. Sequence grabber panel components allow sequence grabber components to obtain configuration information from the user for a particular sequence grabber channel component. Applications never call sequence grabber panel components directly; application developers use panel components only by calling the sequence grabber component.

Managing Your Panel Component

Sequence grabber components load, configure, and unload your panel component. As part of this process, the sequence grabber installs your panel's dialog items into the settings dialog box and may open your component's **resource** file. Panel components provide a number of functions that allow the sequence grabber to manage its relationship with panel components.

After opening a connection to your panel component, the sequence grabber identifies itself to your component by calling your `SGPanelSetGrabber` (III–1767) function. The sequence grabber then tries to determine whether your component can work with its associated channel component by calling your `SGPanelCanRun` (III–1759) function. The sequence grabber calls this function only if you have set the `channelFlagHasDependency` component flag to 1.

Once the sequence grabber has determined that your panel component can work with its channel component, the sequence grabber may open your component's **resource** file (unless you have set the `channelFlagDontOpenResFile` component flag

to 1). Once it has opened the resource file, it passes the file's reference number to you by calling your `SGPanelSetResFile` (III-1768) function.

Next, the sequence grabber prepares to add your component's items to the settings dialog box. The sequence grabber obtains your item list by calling your `SGPanelGetDit1` (III-1761) function. Once it has installed the items, it calls your `SGPanelInstall` (III-1764) function, giving you an opportunity to set initial values.

Before the sequence grabber removes your items from the settings dialog box, it calls your `SGPanelRemove` (III-1766) function.

Functions

`SGPanelSetGrabber` (III-1767)

Identifies a sequence grabber component to a panel component.

`SGPanelCanRun` (III-1759)

Lets a sequence grabber component determine whether a panel component can work with the current sequence grabber channel component.

`SGPanelSetResFile` (III-1768)

Lets the sequence grabber pass a **resource** file's reference number.

`SGPanelGetDit1` (III-1761)

Lets a sequence grabber component determine the dialog items managed by your panel component.

`SGPanelGetTitle` (III-1763)

Gets the displayed title of a sequence grabber panel.

`SGPanelInstall` (III-1764)

Installs added items in a sequence grabber settings dialog box before the dialog box is displayed to the user.

`SGPanelRemove` (III-1766)

Removes a panel from the sequence grabber settings dialog box.

Processing Your Panel's Events

When your panel component is loaded into the settings dialog box and active, you may receive and process dialog events and mouse clicks.

Your component's `SGPanelEvent` (III–1760) function acts like a modal-dialog filter function, allowing you to process individual dialog events. The sequence grabber calls your `SGPanelItem` (III–1765) function whenever the user clicks a dialog item.

Whenever the user clicks the OK button, the sequence grabber calls your `SGPanelValidateInput` (III–1770) function. Your panel component may then validate the user's settings.

Functions

`SGPanelItem` (III–1765)

Receives and processes mouse clicks in the sequence grabber settings dialog box.

`SGPanelEvent` (III–1760)

Lets a component receive and process dialog events.

`SGPanelSetEventFilter` (III–1767)

Sets the event filter callback for a sequence grabber panel component.

`SGPanelValidateInput` (III–1770)

Validates the contents of the user dialog box for a sequence grabber component.

Managing Your Panel's Settings

Sequence grabber components store their configuration information in Movie Toolbox user data items. This configuration information includes settings for each of the channels used by the sequence grabber. Because your panel component configures sequence grabber channels, your panel component is responsible for creating and formatting the contents of its user data items. The sequence grabber component calls your component whenever it wants to retrieve these settings. The sequence grabber may also use previously stored settings to restore your panel's settings.

The sequence grabber calls your `SGPanelGetSettings` (III–1762) function in order to retrieve your panel's current settings. The sequence grabber uses your `SGPanelSetSettings` (III–1769) function to restore those settings to some previous values.

Functions

`SGPanelGetSettings` (III–1762)

Retrieves a panel's current settings for a sequence grabber component.

SGPanelSetSettings (III–1769)

Restores a panel's current settings for a sequence grabber component.

Working With Data Handlers

Data handler components allow QuickTime to retrieve time-based data from external storage devices and, in some cases, store time-based data on those devices. In most cases, you do not need to create a data handler or use one directly, because QuickTime takes care of data storage and retrieval for you through its built-in media handlers. However, you may need to create a data handler component to read or write to a non-Macintosh storage medium.

Selecting a Data Handler

So client programs can choose the best data handler component for a data reference, several functions allow applications to interrogate a data handler's capabilities.

DataHGetVolumeList (I–207) allows an application to obtain a list of the volumes your data handler can support. DataHCanUseDataRef (I–175) allows your data handler to examine a specific data reference and indicate its ability to work with the associated container. DataHGetDeviceIndex (I–193) allows applications to determine whether different data references identify containers that reside on the same device.

Functions

DataHGetVolumeList (I–207)

Returns a list of the volumes your component can access, along with flags indicating your component's capabilities for each volume.

DataHCanUseDataRef (I–175)

Reports whether a data handler can access the data associated with a specified data reference.

DataHGetDeviceIndex (I–193)

Returns a value that identifies the device on which a data reference resides.

Identifying Data References

All data handler components use data references to identify and locate a movie's container. Different types of containers may require different types of data references. Client programs can correlate data references with data handlers by matching the component's subtype value with the data reference type. The subtype value indicates the type of data reference the component supports. All data handlers with the same subtype value must support the same data reference type.

`DataHSetDataRef` (I-224) and `DataHGetDataRef` (I-189) allow applications to assign your data handler's current data reference. `DataHCompareDataRef` (I-178) asks your component to compare a data reference against the current data reference and indicate whether the references are equivalent (that is, refer to the same container). `DataHResolveDataRef` (I-218) permits your component to locate a data reference's container.

Functions

`DataHSetDataRef` (I-224)

Assigns a data reference to your data handler component.

`DataHGetDataRef` (I-189)

Retrieves your component's current data reference.

`DataHCompareDataRef` (I-178)

Compares a supplied data reference against its current data reference and returns a `Boolean` value indicating whether the data references are equivalent (that is, the two data references identify the same container).

`DataHResolveDataRef` (I-218)

Locates the container associated with a given data reference.

Reading Movie Data

Data handler components support several functions that help applications and other components read movie data.

`DataHGetData` (I-186) provides a fully synchronous read operation, while `DataHScheduleData` (I-220) is asynchronous. By calling `DataHFinishData` (I-182), applications can force your component to process queued read requests. Applications may call `DataHGetScheduleAheadTime` (I-206) in order to determine how far in advance your component prefers to get read requests. Before any application

can read data from a data reference, it must open read access to that reference by calling `DataHOpenForRead` (I-209). `DataHCloseForRead` closes that read access path.

Functions

`DataHGetData` (I-186)

Reads data from its current data reference, which is a synchronous read operation.

`DataHScheduleData` (I-220)

Reads data from its current data reference, which can be a synchronous read operation or an asynchronous read operation.

`DataHFinishData` (I-182)

Completes or cancels one or more queued read requests.

`DataHGetScheduleAheadTime` (I-206)

Reports how far in advance it prefers clients to issue read requests.

`DataHOpenForRead` (I-209)

Opens a data handler's current data reference for read-only access.

`DataHCloseForRead` (I-176)

Closes read-only access to its data reference.

Writing Movie Data

As they do with reading movie data, data handlers provide facilities for writing both synchronous and asynchronous data.

`DataHPutData` (I-216) is a simple synchronous interface that allows applications to append data to the end of a container. `DataHWrite` (I-232) is a more capable, asynchronous write function that is suitable for movie capture operations. The component calls the application's data-handler completion function when you are done with the write request.

`DataHCreateFile` (I-180) asks the component to create a new container.

`DataHSetFileSize` (I-227) and `DataHGetFileSize` (I-194) work with a container's size, in bytes. `DataHGetFreeSpace` (I-198) allows applications to determine when to make a container larger. `DataHPreextend` (I-214) asks your component to make a container larger.

Applications may call `DataHGetPreferredBlockSize` (I-204) in order to determine

how best to interact with your data handler. Before writing data to a data reference, applications must call `DataHOpenForWrite` (I–210) to open a write path to the container. `DataHCloseForWrite` (I–177) closes that write path.

Functions

`DataHPutData` (I–216)

Writes data to a component's current data reference.

`DataHWrite` (I–232)

Writes data to its current data reference.

`DataHCreateFile` (I–180)

Creates a new container that meets the specifications of the current data reference.

`DataHSetFileSize` (I–227)

Sets the size, in bytes, of the current data reference.

`DataHGetFileSize` (I–194)

Returns the size, in bytes, of the current data reference.

`DataHGetFreeSpace` (I–198)

Reports the number of bytes available on the device that contains the current data reference.

`DataHPreextend` (I–214)

Allocates new space for the current data reference, enlarging the container.

`DataHGetPreferredBlockSize` (I–204)

Reports the block size that it prefers to use when accessing the current data reference.

`DataHOpenForWrite` (I–210)

Opens your component's current data reference for write-only access.

`DataHCloseForWrite` (I–177)

Closes write-only access to its data reference.

Managing Data Handler Components

Data handler components provide a number of functions that applications can use to manage their connections to them.

The most important among these is `DataHTask` (I–231), which provides processor time to the handler. Applications should call this function often so that the handler has enough time to do its work. Applications may call `DataHPlaybackHints` (I–211) in order to provide the handler with some guidelines about how those applications plan to use the current data reference. `DataHFlushData` (I–184) and `DataHFlushCache` (I–184) allow applications to influence how your component manages its stored data.

Functions

`DataHTask` (I–231)

Cedes processor time to your data handler.

`DataHPlaybackHints` (I–211)

Provides additional information to your component that you may use to optimize the operation of your data handler.

`DataHFlushData` (I–184)

Forces any data in your component's write buffers to be written to the device that contains the current data reference.

`DataHFlushCache` (I–184)

Discards the contents of any cached read buffers.

Using Video Digitizers

Video digitizer components convert a video input into a digitized color image that is compatible with the graphics system of the computer.

Video Digitizer Data Structures

Video digitizer components and applications that use video digitizer components communicate through two primary data structures.

The contents of the `DigitizerInfo` (IV–2234) structure fully define the capabilities and current status of the video digitizer component. You specify video output buffers in a `VdigiBufferRecList` (IV–2499) structure. Note that all the output buffers

must be the same size and must accommodate output rectangles of the same dimensions.

Data Structures

`DigitizerInfo` (IV–2234)

Contains information about the capabilities and current status of a video digitizer component.

`VdigBufferRecList` (IV–2499)

Contains a list of `VdigBufferRec` (IV–2498) structures.

Getting Information About Video Digitizer Components

Two functions allow applications to obtain information about the capabilities and current state of video digitizer components.

You can use `VDGetDigitizerInfo` (III–1998) in your application to retrieve information about the capabilities of a video digitizer component. You can use `VDGetCurrentFlags` (III–1996) to obtain current status information from a video digitizer component.

Functions

`VDGetDigitizerInfo` (III–1998)

Returns capability and status information about a specified video digitizer component.

`VDGetCurrentFlags` (III–1996)

Returns status information about a specified video digitizer component.

Setting Source Characteristics

Several video digitizer component functions allow applications to set the spatial characteristics of the source video signal. You can use these functions in your application to set and retrieve information about the maximum source rectangle, the active source rectangle, the vertical blanking rectangle, and the digitizer rectangle.

You can use `VDGetMaxSrcRect` (III–2012) in your application to get the size and location of the maximum source rectangle. Similarly, `VDGetActiveSrcRect` (III–1989) allows you to get this information about the active source rectangle, and

VDGetVBlankRect (III–2021) enables you to obtain information about the vertical blanking rectangle. You can use VDSaveDigitizerRect (III–2038) to set the size and location of the digitizer rectangle. VDGetDigitizerRect (III–1999) lets you retrieve the size and location of this rectangle.

Functions

VDGetMaxSrcRect (III–2012)

Returns the maximum source rectangle.

VDGetActiveSrcRect (III–1989)

Obtains size and location information for the active source rectangle used by a video digitizer component.

VDGetVBlankRect (III–2021)

Returns the vertical blanking rectangle.

VDSetDigitizerRect (III–2038)

Sets the current video digitizer rectangle.

VDGetDigitizerRect (III–1999)

Returns the current digitizer rectangle.

Selecting an Input Source

Several video digitizer component functions allow applications to select an input video source.

Applications can use VDGetNumberOfInputs (III–2013) to determine the number of video inputs supported by the digitizer component. VDGetInputFormat (III–2005) function allows applications to find out the video format (composite, s-video, or component) employed by a specified input. You can use VDSetInput (III–2042) in your application to specify the input to be used by the digitizer component.

VDGetInput (III–2003) returns the currently selected input. VDSetInputStandard (III–2045) allows you to specify the video signaling standard to be used by the video digitizer component.

Functions

VDGetNumberOfInputs (III–2013)

Returns the number of input video sources that a video digitizer component supports.

`VDGetInputFormat` (III–2005)

Determines the format of the video signal provided by a specified video input source.

`VDSetInput` (III–2042)

Selects the input video source for a video digitizer component.

`VDGetInput` (III–2003)

Returns data that identifies the currently active input video source.

`VDSetInputStandard` (III–2045)

Specifies the input signaling standard to digitize.

Setting Video Destinations

Video digitizer components provide several functions that allow applications to specify the destination for the digitized video stream produced by the digitizer component. Applications have two options for specifying the destination for the video data stream. The first option requires that the video be digitized as RGB pixels and placed into a destination pixel map. This option allows the video to be placed either onscreen or offscreen, depending upon the placement of the pixel map. The second option uses a global boundary rectangle to define the destination for the video. This option is useful only with digitizers that support hardware DMA.

You can use `VDSetPlayThruDestination` (III–2048) in your application to set the characteristics for video digitized as RGB pixels and placed into a destination pixel map. `VDPreflightDestination` (III–2026) lets you determine the capabilities of the digitizer in your application. All video digitizer components must support this option. `VDGetPlayThruDestination` (III–2014) lets you get data about the current video destination.

You can use `VDSetPlayThruGlobalRect` (III–2049) in your application to set the characteristics for a global boundary rectangle defining the destination for the video. You can use `VDPreflightGlobalRect` (III–2028) in your application to determine the capabilities of the digitizer. Not all video digitizer components support this option. `VDGetMaxAuxBuffer` (III–2011) returns information about a buffer that may be located on some special hardware.

Functions

`VDSetPlayThruDestination` (III–2048)

Establishes the destination settings for a video digitizer component.

`VDPreflightDestination` (III–2026)

Verifies that a video digitizer component can support a set of destination settings intended for use with `VDSetPlayThruDestination` (III–2048).

`VDGetPlayThruDestination` (III–2014)

Obtains information about the current video destination.

`VDSetPlayThruGlobalRect` (III–2049)

Establishes the destination settings for a video digitizer component that is to digitize into a global rectangle.

`VDPreflightGlobalRect` (III–2028)

Verifies that a video digitizer component can support a set of destination settings intended for use with `VDSetPlayThruGlobalRect` (III–2049).

`VDGetMaxAuxBuffer` (III–2011)

Obtains access to buffers that are located on special hardware.

Controlling Compressed Source Devices

Some video digitizer components may provide functions that allow applications to work with digitizing devices that can provide compressed image data directly. Such devices allow applications to retrieve compressed image data without using the Image Compression Manager. However, in order to display images from the compressed data stream, there must be an appropriate decompressor component available to decompress the image data. Video digitizers that can support compressed source devices set the `digiOutDoesCompress` flag to 1 in their capability flags.

All of these digitizing functions support only asynchronous digitization. That is, the video digitizer works independently to digitize each frame. Applications are free to perform other work while the digitizer works on each frame. The video digitizer component manages its own buffer pool for use with these functions. In this respect, these functions differ from the other video digitizer functions that support asynchronous digitization.

Applications can use `VDGetCompressionTypes` (III–1994) to determine the image-compression capabilities of a video digitizer. `VDSetCompression` (III–2034) allows applications to set some parameters that govern image compression.

Applications control digitization by calling `VDCompressOneFrameAsync` (III–1988), which instructs the video digitizer to create one frame of compressed image data.

VDCompressDone (III–1986) returns that frame. When an application is done with a frame, it calls VDRestoreCompressBuffer (III–2030) to free the buffer. An application can force the digitizer to place a **key frame** into the sequence by calling VDRestoreCompressSequence (III–2030). Applications can turn compression on and off by calling VDSetCompressionOnOff (III–2035).

Applications can obtain the digitizer’s image description structure by calling VDGetImageDescription (III–2002). Applications can set the digitizer’s time base by calling VDSetTimeBase (III–2054).

Functions

VDGetCompressionTypes (III–1994)

Determines the image-compression capabilities of the video digitizer.

VDGetCompressionTime (III–1993)

Confirms or quantifies a video digitizer’s compression settings.

VDSetCompression (III–2034)

Specifies certain compression parameters.

VDCompressOneFrameAsync (III–1988)

Instructs the video digitizer to digitize and compress a single frame of image data.

VDCompressDone (III–1986)

Determines whether the video digitizer has finished digitizing and compressing a frame of image data.

VDRestoreCompressBuffer (III–2030)

Frees a buffer received from VDCompressDone (III–1986).

VDRestoreCompressSequence (III–2030)

Forces the video digitizer to insert a **key frame** into a temporally compressed image sequence.

VDSetCompressionOnOff (III–2035)

Allows an application to start and stop compression by video digitizers that can deliver either compressed or uncompressed image data.

VDGetImageDescription (III–2002)

Retrieves an ImageDescription (IV–2285) structure from a video digitizer.

VDSetTimeBase (III–2054)

Establishes the video digitizer’s time coordinate system.

`VDSetDataRate` (III–2037)

Instructs your video digitizer component to limit the rate at which it delivers compressed, digitized video data.

`VDGetSoundInputSource` (III–2019)

Instructs your video digitizer component to return the sound input source associated with a particular video input.

Controlling Digitization

Several video digitizer component functions allow applications to control video digitization. Video digitizer components allow applications to start and stop the digitizing process. Your application can request continuous digitization or single-frame digitization. When a digitizer component is operating continuously, it automatically places successive frames of digitized video into the specified destination. When a digitizer component works with a single frame at a time, the application and other software, such as an image compressor component, control the speed at which the digitized video is processed.

You can use `VDSetPlayThruOnOff` (III–2050) in your application to enable or disable digitization. When digitization is enabled, the video digitizer component places digitized video frame into the specified destination continuously. The application stops the digitizer by disabling digitization. This function can be used with both destination options. Alternatively, your application can control digitization on a frame-by-frame basis. `VDGrabOneFrame` (III–2024) and `VDGrabOneFrameAsync` (III–2025) digitize a single video frame. `VDDone` (III–1988) helps you to determine when `VDGrabOneFrameAsync` (III–2025) is finished with a video frame. Your application can define the buffers for use with asynchronous digitization by calling `VDSetupBuffers` (III–2055). Free the buffers by calling `VDReleaseAsyncBuffers` (III–2029). `VDSetFrameRate` allows applications to control the digitizer's frame rate. `VDGetDataRate` (III–1997) returns the digitizer's current data rate.

Functions

`VDSetPlayThruOnOff` (III–2050)

Controls continuous digitization.

`VDGrabOneFrame` (III–2024)

Instructs the video digitizer component to digitize a single frame of source video.

VDGrabOneFrameAsync (III–2025)

Instructs the video digitizer component to start to digitize asynchronously a single frame of source video.

VDDone (III–1988)

Determines if **VDGrabOneFrameAsync (III–2025)** is finished with a specific output buffer.

VDSetupBuffers (III–2055)

Defines output buffers for use with asynchronous grabs.

VDReleaseAsyncBuffers (III–2029)

Releases the buffers that were allocated with **VDSetupBuffers (III–2055)**.

VDSetFrameRate (III–2041)

Indicates an application's desired frame rate to the video digitizer.

VDGetDataRate (III–1997)

Retrieves information that describes the performance capabilities of a video digitizer.

VDSetPreferredPacketSize (III–2052)

Sets the preferred packet size for video digitizing.

VDGetTimeCode (III–2020)

Instructs your video digitizer component to return timecode information for the incoming video signal.

Controlling Color

Video digitizer components support color digitization. Therefore, these components provide several functions that allow applications to control the color digitization process.

You can use **VDSetInputColorSpaceMode (III–2043)** in your application to enable and disable color digitization; you can use **VDGetInputColorSpaceMode (III–2004)** to determine whether color digitization is enabled. **VDUseThisCLUT (III–2057)** allows you to specify a **color lookup table** to be used by the video digitizer component. In cases where the component cannot accommodate a particular lookup table, your application can use **VDGetCLUTInUse (III–1992)** to retrieve the color lookup table used by the digitizer component. Your application can determine a digitizer's supported pixel depths by calling **VDGetDMADepths (III–1999)**.

Functions

- VDSetInputColorSpaceMode (III–2043)
Chooses between color and grayscale digitized video.
- VDGetInputColorSpaceMode (III–2004)
Determines whether a digitizer is operating in color or grayscale mode.
- VDUseThisCLUT (III–2057)
Specifies the lookup table for color digitization.
- VDGetCLUTInUse (III–1992)
Obtains the **color lookup table** used by a video digitizer component.
- VDGetDMADepths (III–1999)
Determines which pixel depths a digitizer supports.

Controlling Analog Video

Some video digitizer components may provide functions that allow applications to control the characteristics of the input analog video signal.

VDGetVideoDefaults (III–2022) returns the suggested default values for the analog video parameters that can be affected by functions described below. You can use VDSetInputGammaValue (III–2044) and VDGetInputGammaValue to allow your application to set particular gamma values. The VDSetBlackLevelValue (III–2031), VDGetBlackLevelValue, VDSetWhiteLevelValue (III–2055), and VDGetWhiteLevelValue functions allow applications to work with black levels and white levels in the source video. VDSetContrast (III–2036), VDGetContrast (III–1995), VDSetSharpness (III–2053), and VDGetSharpness (III–2018) allow applications to work with contrast and sharpness values in the source video. VDGetBrightness (III–1991) and VDSetBrightness (III–2032) allow applications to work with the image brightness setting. VDSetHue (III–2041), VDGetHue (III–2002), VDSetSaturation (III–2053), and VDGetSaturation (III–2017) allow applications to work with hue and saturation settings in the source video.

Functions

- VDGetVideoDefaults (III–2022)
Returns the recommended values for many of the analog video parameters that may be set by applications.
- VDSetInputGammaValue (III–2044)
Sets the gamma values.

`VDGetInputGammaValue` (III–2006)
Returns the current gamma values.

`VDSetBlackLevelValue` (III–2031)
Sets the current black level value.

`VDGetBlackLevelValue` (III–1990)
Returns the current black level value.

`VDSetWhiteLevelValue` (III–2055)
Sets the white level value.

`VDGetWhiteLevelValue` (III–2024)
Returns the current white level value.

`VDSetContrast` (III–2036)
Sets the current contrast value.

`VDGetContrast` (III–1995)
Returns the current contrast value.

`VDSetSharpness` (III–2053)
Sets the sharpness value.

`VDGetSharpness` (III–2018)
Returns the current sharpness value.

`VDGetBrightness` (III–1991)
Returns the current brightness value.

`VDSetBrightness` (III–2032)
Sets the current brightness value.

`VDSetHue` (III–2041)
Sets the current hue value.

`VDGetHue` (III–2002)
Returns the current hue value.

`VDSetSaturation` (III–2053)
Sets the saturation value.

`VDGetSaturation` (III–2017)
Returns the current saturation value.

Selectively Displaying Video

Video digitizer components may support one of three methods of selectively displaying video: key colors, alpha channels, and blend masks.

Your applications can use `VDSetKeyColor` (III–2046), `VDAddKeyColor` (III–1985), and `VDSetKeyColorRange` (III–2046) to set one or more key colors for a video digitizer component. `VDGetKeyColor` (III–2008), `VDGetNextKeyColor` (III–2013), and `VDGetKeyColorRange` (III–2009) allow your application to retrieve information about the currently active key colors. Your applications can use `VDGetMaskandValue` (III–2009) to determine the appropriate mask value for a desired blend level. `VDSetMasterBlendLevel` (III–2047) allows applications to set a blend level that applies to the entire source video image. `VDGetMaskPixMap` (III–2011) allows applications to retrieve the pixel map that defines the blend mask.

Functions

`VDSetKeyColor` (III–2046)

Sets the key color for video digitizing.

`VDAddKeyColor` (III–1985)

Adds a key color to a component's list of active key colors.

`VDSetKeyColorRange` (III–2046)

Defines a key color range for video digitizing.

`VDGetKeyColor` (III–2008)

Obtains the index value of the active key color.

`VDGetNextKeyColor` (III–2013)

Obtains the index value of the active key colors in cases where the digitizer component supports multiple key colors.

`VDGetKeyColorRange` (III–2009)

Obtains the currently defined key color range.

`VDGetMaskandValue` (III–2009)

Obtains the appropriate alpha channel or blend mask value for a desired level of video blending.

`VDSetMasterBlendLevel` (III–2047)

Sets the blend level value for the input video signal.

`VDGetMaskPixMap` (III–2011)

Retrieves the pixel map data for a component's blend mask.

Video Clipping

Some video digitizer components can clip the output video image based on an arbitrary clipping region. The functions that manipulate clipping and clipping state operate on a clipping region in addition to the one specified by the mask passed by `VDSetPlayThruDestination` (III–2048) and `VDSetUpBuffers`. To determine the final clipping regions, intersect these two clippings.

Applications can use `VDSetClipState` (III–2033) and `VDGetClipState` (III–1991) to enable and disable clipping, and to determine whether clipping is enabled. Applications can use `VDSetClipRgn` (III–2032) and `VDClearClipRgn` (III–1986) to manipulate the clipping region. Applications can use these functions only during an active grab sequence. Applications set the initial clipping settings by calling either `VDSetPlayThruDestination` (III–2048) or `VDSetPlayThruGlobalRect`.

Functions

`VDSetClipState` (III–2033)

Controls whether clipping is enabled.

`VDGetClipState` (III–1991)

Determines whether clipping is enabled.

`VDSetClipRgn` (III–2032)

Defines a clipping region for a video digitizer.

`VDClearClipRgn` (III–1986)

Disables all or part of a clipping region that was previously set with `VDSetClipRgn` (III–2032).

`VDSetPlayThruDestination` (III–2048)

Establishes the destination settings for a video digitizer component.

`VDSetPlayThruGlobalRect` (III–2049)

Establishes the destination settings for a video digitizer component that is to digitize into a global rectangle.

See Also

See “Setting Video Destinations” (V–2845).

Video Digitizer Utilities

Certain utility functions may be supported by some video digitizer components.

VDSetPLLFilterType (III–2051) and VDGetPLLFilterType allow applications to control which phase-locked loop (PLL) is used by a video digitizer component that supports multiple PLLs. VDSetFieldPreference (III–2040) and VDGetFieldPreference allow applications to control which field is used for some vertical scaling operations. VDSetDigitizerUserInterrupt (III–2039) allows applications to install custom interrupt functions that are called by the video digitizer component. VDGetSoundInputDriver (III–2019) allows an application to retrieve information about a digitizer’s sound input driver. VDGetPreferredTimeScale (III–2017) allows an application to determine a digitizer’s preferred time scale.

Applications can provide a custom interrupt function by calling VDSetDigitizerUserInterrupt (III–2039).

Functions

VDSetPLLFilterType (III–2051)

Specifies which phase locked loop (PLL) is to be active.

VDGetPLLFilterType (III–2015)

Determines which phase locked loop (PLL) mode is currently active for a video digitizer.

VDSetFieldPreference (III–2040)

Specifies which field to use in cases where the vertical scaling is less than half size.

VDGetFieldPreference (III–2001)

Determines which field is being used in cases where the image is vertically scaled to half its original size.

VDSetDigitizerUserInterrupt (III–2039)

Sets custom interrupt functions.

VDGetSoundInputDriver (III–2019)

Retrieves information about a video digitizer’s sound input driver.

VDGetPreferredTimeScale (III–2017)

Determines a digitizer’s preferred time scale.

VDSetDigitizerUserInterrupt (III–2039)

Sets custom interrupt functions.

Callback

VdigIntProc (III–2154)

An interrupt callback called by the video digitizer component each time it starts to display a field.

Exchanging Movie Data

Movie data exchange components allow applications to place various types of data into a QuickTime movie or extract data from a movie in a specified format.

Importing Movie Data

Movie data import components may provide functions that allow the Movie Toolbox to request a data conversion operation. Before the Movie Toolbox calls one of these functions, a requesting application may call one or more of your component's configuration functions. However, your component should work properly even if none of these configuration functions is called.

MovieImportHandle (II–950) instructs your component to retrieve the data that is to be imported from a specified handle. MovieImportFile (II–942) instructs you to retrieve the data from a file. You should set the appropriate flags in your component's `componentFlags` field to indicate which of these functions your component supports. Note that your component may support both functions. Other functions are listed below.

Functions

MovieImportHandle (II–950)

Imports data from a handle, using a movie data import component.

MovieImportFile (II–942)

Imports data from a file, using a movie data import component.

MovieImportGetFileType (II–946)

Allows your movie data import component to tell the Movie Toolbox the appropriate file type for the most-recently imported movie file.

`MovieImportGetAuxiliaryDataType` (II–944)

Returns the type of the auxiliary data that a component can accept.

`MovieImportGetMIMETYPEList` (II–948)

Returns a list of MIME types supported by the movie import component.

`MovieImportValidate` (II–964)

Allows your movie data import component to validate the data to be passed to your component.

`MovieImportValidateDataRef` (II–965)

Validates the data file indicated by the data reference.

`MovieImportSetOffsetAndLimit` (II–959)

Specifies location and size of data that should be imported.

`MovieImportSetOffsetAndLimit64` (II–960)

Specifies location and size of data that should be imported from a file.

`MovieImportGetSettingsAsAtomContainer` (II–949)

Retrieves the current settings from the movie import component.

`MovieImportSetSettingsFromAtomContainer` (II–963)

Sets the movie import component’s current configuration from the passed settings data.

See Also

See also “Using Graphics Importers” (V–2876).

Configuring Movie Data Import Components

Your component may provide one or more configuration functions that allow applications to configure your component before the Movie Toolbox calls your component to start the import process. Note that applications may call these functions directly. All of these functions are optional. If your component receives a request that it does not support, you should return the `badComponentSelector` error code. In addition, your component should work properly even if none of these functions is called.

`MovieImportSetSampleDuration` (II–962) allows an application to set your component’s sample duration. Use `MovieImportSetDuration` (II–957) to control the duration of the imported data. Applications can use `MovieImportSetDimensions` (II–955) to specify the spatial dimensions of a new track. Use `MovieImportSetSampleDescription` (II–961) to supply a sample description structure

to your movie data import component.

`MovieImportSetMediaFile` (II–958) allows applications to direct your component's output to a specific media file. Applications can provide additional data to your component by calling `MovieImportSetAuxiliaryData` (II–954).

`MovieImportSetChunkSize` allows applications to control the chunk size in the new media. Applications can inform you that the source data came from the scrap by calling `MovieImportSetFromScrap` (II–957). Applications can specify a **progress function** for use by your component by calling `MovieImportSetProgressProc` (II–961). Applications can instruct your component to display its user dialog box by calling `MovieImportDoUserDialog` (II–940).

Functions

`MovieImportSetSampleDuration` (II–962)

Sets the sample duration for new samples to be created with a component.

`MovieImportSetDuration` (II–957)

Controls the duration of the data that a component pastes into the target movie.

`MovieImportSetDimensions` (II–955)

Specifies a new track's spatial dimensions.

`MovieImportSetSampleDescription` (II–961)

Provides a `SampleDescription` (IV–2414) structure to a movie data import component.

`MovieImportSetMediaFile` (II–958)

Specifies a media file that is to receive the imported movie data.

`MovieImportSetAuxiliaryData` (II–954)

Provides additional data to a component.

`MovieImportSetChunkSize` (II–954)

The amount of data a component works with at a time.

`MovieImportSetFromScrap` (II–957)

Indicates that the source data resides on the scrap.

`MovieImportSetProgressProc` (II–961)

Assigns a movie **progress function**.

`MovieImportDoUserDialog` (II–940)

Requests that a component display its user dialog box.

Exporting Movie Data

Movie data export components may provide one or two functions that allow the Movie Toolbox to request a data conversion operation. Before the Movie Toolbox calls one of these functions, a requesting application may call one or more of your component's configuration functions. However, your component should work properly even if none of these configuration functions is called.

`MovieExportToHandle` (II-936) instructs your component to place the converted data into a specified handle. `MovieExportToFile` (II-934) instructs you to put the data into a file. You should set the appropriate flags in your component's `componentFlags` field to indicate which of these functions your component supports. Note that your component may support both functions. Additional functions are listed below.

Functions

`MovieExportToHandle` (II-936)

Exports data from a movie, using a movie data export component.

`MovieExportToFile` (II-934)

Exports data to a file, using a movie data export component.

`MovieExportGetAuxiliaryData` (II-923)

Retrieves additional data from a component.

`MovieExportSetSampleDescription` (II-931)

Requests the format of the exported data.

`MovieExportValidate` (II-937)

Determines whether a movie export component can export all the data for a specified movie or track.

`MovieExportAddDataSource` (II-919)

Defines a data source for use with an export operation performed by `MovieExportFromProceduresToDataRef` (II-922).

`MovieExportFromProceduresToDataRef` (II-922)

Exports data provided by `MovieExportAddDataSource` (II-919) to a specified location.

`MovieExportSetGetMoviePropertyProc` (II-929)

Specifies the procedure that the export component should call to retrieve movie level properties during `MovieExportFromProceduresToDataRef` (II-922).

MovieExportToDataRef (II-933)

Allows an application to request that data be exported to a data reference instead of to a file.

MovieExportGetSettingsAsAtomContainer (II-925)

Retrieves the current settings from the movie export component.

MovieExportSetSettingsFromAtomContainer (II-932)

Sets the movie export component's current configuration from passed settings data.

MovieExportNewGetDataAndPropertiesProcs (II-927)

Returns MovieExportGetPropertyProc (III-2102) and MovieExportGetDataProc (III-2101) callbacks that can be passed to MovieExportAddDataSource (II-919) to create a new data source.

MovieExportDisposeGetDataAndPropertiesProcs (II-920)

Disposes of the memory associated with the procedures returned by MovieExportNewGetDataAndPropertiesProcs (II-927).

See Also

See also "Using Graphics Exporters" (V-2883).

Configuring Movie Data Export Components

Your component may provide one or more configuration functions. These functions allow applications to configure your component before the Movie Toolbox calls your component to start the export process. Note that applications may call these functions directly. All of these functions are optional. If your component receives a request that it does not support, you should return the badComponentSelector error code. In addition, your component should work properly even if none of these functions is called.

Applications can retrieve additional data from your component by calling MovieExportGetAuxiliaryData (II-923). Applications can specify a **progress function** for use by your component by calling MovieExportSetProgressProc (II-930). Applications can instruct your component to display its user dialog box by calling MovieExportDoUserDialog (II-921).

Functions

MovieExportGetAuxiliaryData (II-923)

Retrieves additional data from a component.

MovieExportSetProgressProc (II–930)

Assigns a movie **progress function**.

MovieExportDoUserDialog (II–921)

Requests that a component display its user dialog box.

Exporting Text

The text export and import components provide make it easy to work with the data in a text track in a QuickTime movie. Text descriptors are formatting commands that you can embed within a text file. Time stamps describe a text sample's starting time and duration. When you export text from a text track, you can optionally export text descriptors and time stamps for the text. You can open the text file in a word processor and make changes to the text, style, color, and time stamps. You can then import the edited text to a text track where all the timing, style, color and time stamp information will be present.

The functions listed below support text exporting.

Functions

TextExportGetDisplayData (III–1928)

Retrieves text display information for the current sample in the specified text export component.

TextExportGetTimeFraction (III–1929)

Retrieves the time scale the specified text export component uses to calculate time stamps.

TextExportSetTimeFraction (III–1931)

Sets the time scale the specified text export component uses to calculate time stamps.

TextExportGetSettings (III–1928)

Retrieves the value of the text export option for the specified text export component.

TextExportSetSettings (III–1930)

Sets the value of the text export option for the specified text export component.

Using Derived Media Handlers

Derived media handler components allow the Movie Toolbox to play the data in a media. These components isolate the Movie Toolbox from the details of how or where a particular media is stored. This not only frees the Movie Toolbox from reading and writing media data, but also makes QuickTime extensible to new data formats. These components are referred to as derived components because they rely on the services of a common base media handler component, which is supplied by Apple. The base media handler component handles most of the duties that must be performed by all media handlers. Your derived media handler component extends the services provided by the base media handler.

Managing Your Media Handler Component

Derived media handlers provide three functions that allow the Movie Toolbox to manage its relationship with the media handler.

The Movie Toolbox calls `MediaInitialize` (II-877) in order to give you an opportunity to prepare to provide access to your media. The Movie Toolbox grants processing time to your handler by calling `MediaIdle` (II-874). Calling `MediaGGetStatus` (II-869) then allows the Movie Toolbox to retrieve status information.

Functions

`MediaInitialize` (II-877)

Prepares a derived media handler component to provide access to its media.

`MediaIdle` (II-874)

Provides processing time to a derived media handler during movie playback.

`MediaGGetStatus` (II-869)

Reports error conditions to the Movie Toolbox.

General Data Management

While the base media handler isolates your component from the details of media data access, your derived media handler still needs to keep track of certain information in order to support movie playback and creation.

Your media handler may store proprietary information in its media. The Movie Toolbox calls two media handler functions in order to give you an opportunity to retrieve or store this information. `MediaPutMediaInfo` (II–884) allows you to store your special information in your media. `MediaGetMediaInfo` (II–853) delivers that data to your media handler.

The Movie Toolbox tells your media handler when its track has been enabled or disabled by calling `MediaSetActive` (II–889). The Movie Toolbox prepares your handler for playback by calling `MediaPreroll` (II–883). Whenever your media's playback rate changes, the Movie Toolbox calls `MediaSetRate` (II–903). Whenever the track that uses your media is edited, the Movie Toolbox calls `MediaTrackEdited` (II–914).

If the Movie Toolbox has called `SetMediaSampleDescription` (III–1606) on a sample description, it uses `MediaSampleDescriptionChanged` (II–887) to notify your media handler of the change.

The Movie Toolbox allows tracks to be identified by various characteristics. For instance, it is possible to request that all tracks containing audio information be searched. To determine whether a track has a given characteristic, the Movie Toolbox queries the media handler for each track. The Movie Toolbox calls `MediaHasCharacteristic` (II–872) with the specified characteristic.

The Movie Toolbox uses two functions to inform you about changes to your media's time environment. `MediaSetMediaTimeScale` (II–898) allows the Movie Toolbox to change your media's time scale. `MediaSetMovieTimeScale` (II–899) allows the Movie Toolbox to tell you when the movie's time scale has changed. Other functions are listed below.

Functions

`MediaPutMediaInfo` (II–884)

Lets a derived media handler store proprietary information in its media.

`MediaGetMediaInfo` (II–853)

Lets a derived media handler obtain the private data stored in its media.

`MediaSetActive` (II–889)

Enables and disables media.

`MediaPreroll` (II–883)

Prepares a media handler for playback.

`MediaSetRate` (II–903)

Sets a media's playback rate.

`MediaTrackEdited` (II–914)

Notifies a derived media handler about edits to its track.

`MediaSampleDescriptionChanged` (II–887)

Notifies a media handler that `SetMediaSampleDescription` (III–1606) has been called for a specified sample description.

`MediaHasCharacteristic` (II–872)

Called by Movie Toolbox with a specified characteristic to allow tracks to be identified by various attributes.

`MediaSetMediaTimeScale` (II–898)

Notifies a media handler that its media's time scale has been changed.

`MediaSetMovieTimeScale` (II–899)

Notifies a media handler that the movie's time scale has been changed.

`MediaGSetActiveSegment` (II–870)

Notifies your derived media handlers of the current active segment.

`MediaInvalidateRegion` (II–879)

Updates the invalidated display region the next time `MediaIdle` is called.

`MediaGetNextStepTime` (II–856)

Searches for the next forward or backward step time from the given media time.

`MediaTrackReferencesChanged` (II–915)

Notifies the derived media handler whenever the track references in the movie change.

`MediaTrackPropertyAtomChanged` (II–915)

Notifies the derived media handler whenever its media property atom has changed.

`MediaSetTrackInputMapReference` (II–908)

Provides a derived media handler with an updated input map.

`MediaGetSampleDataPointer` (II–859)

Allows a derived media handler to obtain a pointer to the sample data for a particular sample number, the size of that sample, and the index of the sample description associated with that sample.

MediaReleaseSampleDataPointer (II–886)

Balances calls to **MediaGetSampleDataPointer (II–859)** to release allocated memory.

MediaCompare (II–844)

Lets a media handler determine whether the Movie Toolbox should allow one track to be pasted into another.

MediaSetVideoParam (II–910)

Lets you dynamically adjust the brightness, contrast, hue, sharpness, saturation, black level, and white level of a video image.

MediaGetVideoParam (II–868)

Retrieves the value of the brightness, contrast, hue, sharpness, saturation, black level, or white level of a video image.

MediaSetNonPrimarySourceData (II–899)

Allows a media handler to support receiving media data from other media handlers.

MediaGetOffscreenBufferSize (II–858)

Determines the dimensions of the offscreen buffer.

MediaSetHints (II–896)

Implements the appropriate behavior for the various media hints such as scrub mode and high-quality mode.

MediaGetName (II–855)

Returns the name of the media type.

Managing Graphics Data

If your media handler draws media data on the screen, you need to manage your media's graphics environment. The Movie Toolbox uses a number of functions to inform you about changes to the graphics environment. The Movie Toolbox only calls these functions if you have set the `handlerHasSpatial` flag to 1 when calling **MediaSetHandlerCapabilities (II–894)**.

The Movie Toolbox calls **MediaSetGWorld (II–893)** whenever your media's graphics port or graphics device has changed. **MediaSetDimensions (II–891)** allows the Movie Toolbox to inform your handler about changes to its spatial dimensions. Whenever either the movie or track matrix changes, the Movie Toolbox calls **MediaSetMatrix (II–897)**. Similarly, if your media's clipping region changes, the Movie Toolbox calls

`MediaSetClip` (II-890).

When it is building up a movie's image from its component tracks, the Movie Toolbox must be able to determine which tracks are transparent. The Movie Toolbox calls `MediaGetTrackOpaque` (II-865) to retrieve this information about your media.

The Movie Toolbox calls `MediaGetNextBoundsChange` (II-855) so that it can learn when your media will next change its display shape. When the Movie Toolbox wants to find out the shape of the region into which you draw your media, it calls `MediaGetSrcRgn` (II-865). Other functions are listed below.

Functions

`MediaSetGWorld` (II-893)

Lets a derived media handler learn about changes to its media's graphic environment.

`MediaSetDimensions` (II-891)

Informs a media handler when its media's spatial dimensions change.

`MediaSetMatrix` (II-897)

Tells a media handler about changes to either the movie matrix or the track matrix.

`MediaSetClip` (II-890)

Specifies changes to a derived media handler's clipping region.

`MediaGetTrackOpaque` (II-865)

Determines whether a media is transparent or opaque when displayed.

`MediaGetNextBoundsChange` (II-855)

Determines when a media causes a spatial change to a movie.

`MediaGetSrcRgn` (II-865)

Specifies an irregular destination display region to the Movie Toolbox.

`MediaGetDrawingRgn` (II-849)

Specifies a portion of the screen that must be redrawn, defined in the movie's display coordinate system.

`MediaGetGraphicsMode` (II-852)

Obtains the graphics mode and blend color values currently in use by any media handler.

`MediaSetGraphicsMode` (II–892)

Sets the graphics mode and blend color of any media handler.

Managing Sound Localization

If you are creating a media handler that plays sound and wish to support 3D sound capabilities, you need to implement the function listed below.

The required function is `MediaSetSoundLocalizationData` (II–907).

Function

`MediaSetSoundLocalizationData` (II–907)

Supports 3D sound capabilities in a media handler that plays sound.

Making a Progressive Download Time Table

When an application or other software calls any of the Movie Toolbox’s functions to create a time table for progressive downloads, the base media handler calls your derived media handler to create the time table

The required function is `MediaMakeMediaTimeTable` (II–879).

Function

`MediaMakeMediaTimeTable` (II–879)

Called by the base media handler to create a media time table.

Reporting Capabilities to the Base Media Handler

Apple’s base media handler component provides a utility function that allows you to tell the base handler what your derived handler can do.

This function, `MediaSetHandlerCapabilities` (II–894), is listed below.

Function

`MediaSetHandlerCapabilities` (II–894)

Lets a derived media handler report its capabilities to the base media handler.

Using Clock Components

Clock components provide two basic services: they generate time information and schedule time-based **callback events**. In QuickTime, the Movie Toolbox is the primary user of clock components. Specifically, the Movie Toolbox uses clock components to provide basic timing to time bases. In general, clock components derive their timing information from some external source.

Getting the Current Time

Clock components provide a single function that allows the Movie Toolbox to obtain the current time.

This function is listed below.

Function

`ClockGetTime` (I-95)

Obtains the current time according to a specified clock.

Using Callback Functions

Applications that use QuickTime time bases may define callback functions that are associated with a specific time base. Applications can then use these callback functions to perform activities that are triggered by temporal events, such as a certain time being reached or a specified rate being achieved. The time base functions of the Movie Toolbox interact with clock components to schedule the invocation of these callback functions, and your clock component is responsible for calling the callback function at its scheduled time.

`ClockNewCallback` (I-96) allows your clock component to allocate the memory to support a new **callback event**. When an application discards a callback event, the Movie Toolbox calls `ClockDisposeCallback` (I-94). The Movie Toolbox calls `ClockCallMeWhen` (I-91) when an application wants to schedule a callback event. When the callback function is to be invoked to service the event, the Movie Toolbox calls `ClockCancelCallback` (I-93) so that you can remove the callback event from the list of scheduled events.

Functions

`ClockNewCallback` (I–96)

In a clock component, allocates memory for a new **callback event**.

`ClockDisposeCallback` (I–94)

In a clock component, disposes of the memory associated with the specified **callback event**.

`ClockCallMeWhen` (I–91)

In a clock component, schedules a **callback event** for invocation.

`ClockCancelCallback` (I–93)

In a clock component, removes the specified **callback event** from the list of scheduled callback events for a time base.

Managing the Time

Clock components provide several functions that allow the Movie Toolbox to alert your component to changes in its environment.

Three of these functions, `ClockTimeChanged` (I–100), `ClockRateChanged`, and `ClockStartStopChanged` (I–99), are associated with application callback functions and help your component determine whether to invoke the callback function. The fourth, `ClockSetTimeBase` (I–98), tells your clock component about the time base it is supporting.

Functions

`ClockTimeChanged` (I–100)

In a clock component, is called whenever the callback's time base time value is set.

`ClockRateChanged` (I–97)

In a clock component, is called whenever the callback's time base rate changes.

`ClockStartStopChanged` (I–99)

In a clock component, is called whenever the start or stop time of the callback's time base changes.

`ClockSetTimeBase` (I–98)

In a clock component, is called when an application creates a time base that uses the clock component.

Movie Toolbox Clock Support Functions

The Movie Toolbox provides a number of support functions for clock components. All of these functions help your component manage its associated callback functions. Your clock component may call any of these functions at interrupt time. These functions should only be called by clock components.

Use `AddCallbackToTimeBase` (I–21) to add a **callback event** to the list of scheduled callback events maintained by the Movie Toolbox. You should use `RemoveCallbackFromTimeBase` (III–1456) to remove a callback event from the list. When your clock component determines that it is time to invoke a callback function, you should use `ExecuteCallback` (I–339) to cause the Movie Toolbox to call the function. If your clock component needs to scan all its associated callback events, you can use `GetFirstCallback` (I–411) and `GetNextCallback` (I–501).

Functions

`AddCallbackToTimeBase` (I–21)

Places a **callback event** into the list of scheduled callback events.

`RemoveCallbackFromTimeBase` (III–1456)

Removes a **callback event** from the list of scheduled callback events.

`ExecuteCallback` (I–339)

Called by a clock component when it determines that it is time to execute a callback function.

`GetFirstCallback` (I–411)

Returns the first **callback event** associated with a specified time base.

`GetNextCallback` (I–501)

Returns the next **callback event** associated with a specified time base.

Using the Timecode Media Handler

The timecode media handler allows QuickTime movies to store timing information that is not created by or for QuickTime. This additional timing information would typically be derived from the original source material (for example, as a SMPTE timecode). In essence, you can think of the timecode media handler as providing a link between the digital QuickTime-specific timing information and the original

analog timing information from the source material.

A movie's timecode is stored in a timecode track. Timecode tracks contain source identification information, timecode format information, and frame numbers. Apple Computer has defined the information that is stored in the track in a manner that is independent of any specific timecode standard. The format of this information is sufficiently flexible to accommodate all known timecode standards, including SMPTE timecoding.

One key timecode attribute relates to the dropframe technique used to synchronize timecode values with video frames. There is a flag in the timecode definition structure that indicates whether the timecode uses the dropframe technique.

Working With the Timecode Media Handler

You can create a timecode track and media in the same manner that you create any other track. Call `NewMovieTrack` (II–1092) to create the timecode track, and use `NewTrackMedia` (II–1120) to create the track's media. Be sure to specify a media type value of `TimeCodeMediaType` when you call `NewTrackMedia` (II–1120).

You can make the toolbox display the timecode when a movie is played. Use `TCS setTimeCodeFlags` (III–1923) to turn the timecode display on and off. Note that the timecode track must be enabled for this display to work.

You can define the relationship between a timecode track and one or more movie tracks using track reference functions; see “Working With Track References” (V–2737). You then add samples to the track as appropriate. Each sample in the timecode track provides timecode information for a span of movie time. The sample includes duration information. As a result, you typically add each timecode sample after you have created the corresponding content track or tracks.

The `TimeCodeDescription` (IV–2483) structure contains control information that allows QuickTime to interpret the samples. This includes the timecode format information. The actual sample data contains a frame number that identifies one or more content frames that use this timecode. This value identifies the first frame in the group of frames that use this timecode. In the case of a movie made from source material that contains no edits, you would only need one sample. When the source material contains edits, you typically need one sample for each edit, so that QuickTime can resynchronize the timecode information with the movie. Those

samples contain the frame numbers of the frames that begin each new group of frames.

Functions

TCSetTimeCodeFlags (III–1923)

Changes the flag that affects how the toolbox handles timecode information.

TCGetTimeCodeFlags (III–1921)

Retrieves the timecode control flags.

TCGetCurrentTimeCode (III–1917)

Retrieves the timecode and source identification information for the current movie time.

TCGetTimeCodeAtTime (III–1920)

Returns a track's timecode information corresponding to a specific media time.

TCGetDisplayOptions (III–1918)

Retrieves the text characteristics that apply to timecode information displayed in a movie.

TCSetDisplayOptions (III–1922)

Sets the text characteristics that apply to timecode information displayed in a movie.

TCGetSourceRef (III–1919)

Retrieves the source information from the timecode media sample reference.

TCSetSourceRef (III–1922)

Changes the source information in the timecode media sample reference.

TCFrameNumberToTimeCode (III–1916)

Converts a frame number into its corresponding timecode time value.

TCTimeCodeToFrameNumber (III–1924)

Converts a timecode time value into its corresponding frame number.

TCTimeCodeToString (III–1925)

Converts a time value into a text string (HH:MM:SS:FF).

Data Structures

TimeCodeDescription (IV–2483)

Defines the format and content of a timecode media sample description.

TimeCodeTime (IV–2485)

Provides time information for a TimeCodeRecord (IV–2484) structure.

Using Preview Components

Preview components are used by the Image Compression Manager’s standard file **preview** functions to display and create visual previews for files. Previews usually consist of a single image, but they may contain many kinds of data, including sound. The Image Compression Manager is the primary client of preview components. Rarely, if ever, do applications call preview components directly.

Displaying Previews

The **preview** component supplies a single function for displaying movie previews.

If your **preview** component does not handle events (that is, does not contain time-based data), you should use PreviewShowData (II–1146).

Functions

PreviewShowData (II–1146)

Displays a **preview** if it does not handle events.

Handling Preview Events

A function is provided so that your **preview** component can do standard event filtering.

If your **preview** component handles events, PreviewEvent (II–1144) is called as appropriate.

Functions

PreviewEvent (II–1144)

May be called as appropriate if a **preview** component handles events.

Creating Previews

Two functions are available for use in creating **previews**.

`PreviewMakePreview` (II–1145) creates **previews** by allocating a handle to data to be added to the file. On the other hand, `PreviewMakePreviewReference` (II–1145) makes previews by returning the type and identification number of a **resource** within the file to be used as the preview for the file.

Functions

`PreviewMakePreview` (II–1145)

Creates **previews** by allocating a handle to data that is to be added to a file.

`PreviewMakePreviewReference` (II–1145)

Returns the type and identification number of a **resource** within a file to be used as the **preview** for a file.

Transcoding Images

For some types of compressed image data, it is possible to convert directly from one compressed format to another. This direct conversion process is called image transcoding. Transcoding has two distinct advantages over the decompress-then-recompress approach to converting the format of compressed data. The first advantage is that the operation is usually substantially faster, since much of the data can be copied directly from the source image data format to the destination image data format. The second advantage is that the operation is usually more accurate because decompressing and recompressing provides two steps for introducing rounding and quantization errors. By directly transcoding, opportunities for small errors are substantially reduced.

Image Transcoder Support

QuickTime provides several image transcoder component functions.

These functions are listed below.

Functions

- ImageTranscoderBeginSequence (II-751)
Initiates an image transcoding sequence and specifies the input data format.
- ImageTranscodeSequenceBegin (II-754)
Initiates an image transcoder sequence operation.
- ImageTranscoderConvert (II-752)
Performs image transcoding operations.
- ImageTranscodeFrame (II-750)
Transcodes a frame of image data.
- ImageTranscoderDisposeData (II-753)
Disposes of transcoded data.
- ImageTranscodeDisposeFrameData (II-749)
Disposes transcoded image data.
- ImageTranscoderEndSequence (II-754)
Ends an image transcoding sequence.
- ImageTranscodeSequenceEnd (II-755)
Ends an image transcoder sequence operation.

Using Text Channel Components

Just as video digitizer components obtain digitized video from an analog video source, text digitizer components obtain text from an external source. A text channel component is a sequence grabber channel component. A text digitizer is a new kind of component. The text channel component uses the services of text digitizer components.

Text Channel Support

The text channel component provides several functions for formatting text to be **previewed** or added to a text track of a movie.

These functions are listed below.

Functions

`SGSetFontName` (III–1790)

Sets the name of the font to be used to display text for a text channel component.

`SGSetFontSize` (III–1791)

Sets the font size to be used to display text for a text channel component.

`SGSetTextForeColor` (III–1807)

Sets the color to be used to display text.

`SGSetTextBackColor` (III–1806)

Sets the background color to be used for the text box.

`SGSetTextJustification` (III–1795)

Sets the alignment to be used to display text for a text channel component.

`SGSetTextReturnToSpaceValue` (III–1737)

Indicates whether the text channel component should replace return characters with spaces.

`SGSetTextReturnToSpaceValue` (III–1807)

Determines whether the text channel component should replace return characters with spaces.

See Also

See “Sequence Grabbing” (V–2808).

Tweening

A **tween** media handler lets you use tween data stored in a tween track to modify the presentation of other tracks algorithmically. When a movie includes a tween track, the tween media handler invokes the tween component (or native QuickTime code) needed to process the tween data and delivers the resulting values to one or more other media handlers.

A tween component algorithmically generates an output value for any point in a time interval. The input for a tween is a small number of values, often as few as one or two, from which a range of values can be derived.

Tween Component Requirements

You must provide certain functions to implement a **tween** component. QuickTime calls these functions when it uses your component.

These functions are listed below.

Functions

`TweenerInitialize` (III–1977)

Initializes your **tween** component for a single tween operation.

`TweenerReset` (III–1978)

Cleans up when the **tween** operation is finished.

`TweenerDoTween` (III–1976)

Performs a **tween** operation.

Using Graphics Importers

Graphics importer components provide a standard method for opening and displaying still images contained within graphics documents and for working with any type of image data, regardless of the file format or compression used in the graphics document.

Managing Graphis Importers

QuickTime provides several functions to manage graphics import components.

These functions are listed below.

Functions

`GraphicsImportGetImageCount` (I–637)

Returns the number of images in an imported image file.

`GraphicsImportSetImageIndex` (I–661)

Specifies the image index for an imported image.

`GraphicsImportGetImageIndex` (I-638)

Returns the current image index for an imported image.

`GraphicsImportGetDataOffsetAndSize` (I-624)

Returns the offset and size of the compressed image data within an imported image file.

`GraphicsImportGetDataOffsetAndSize64` (I-625)

Provides a 64-bit version of `GraphicsImportGetDataOffsetAndSize`.

`GraphicsImportReadData` (I-645)

Reads imported image data.

`GraphicsImportReadData64` (I-646)

Provides a 64-bit version of `GraphicsImportReadData` (I-645).

`GraphicsImportSetDataReferenceOffsetAndLimit` (I-654)

Specifies the data reference starting offset and data size limit for an imported image.

`GraphicsImportSetDataReferenceOffsetAndLimit64` (I-656)

Provides a 64-bit version of `GraphicsImportSetDataReferenceOffsetAndLimit` (I-654).

`GraphicsImportGetDataReferenceOffsetAndLimit` (I-627)

Returns the data reference starting offset and data size limit for an imported image.

`GraphicsImportGetDataReferenceOffsetAndLimit64` (I-628)

Provides a 64-bit version of `GraphicsImportGetDataReferenceOffsetAndLimit` (I-627).

`GraphicsImportGetDefaultMatrix` (I-630)

Returns the default matrix for an imported image, if one is stored there.

`GraphicsImportGetDefaultClip` (I-629)

Returns the default clipping region for an imported image, if one is stored there.

`GraphicsImportGetDefaultGraphicsMode` (I-630)

Returns the default graphics mode for an imported image, if one is stored there.

`GraphicsImportGetDefaultSourceRect` (I-631)

Returns the default source rectangle for an imported image, if one is stored there.

GraphicsImportGetColorSyncProfile (I-621)

Returns a ColorSync profile for an imported image, if one is embedded in the image file.

GraphicsImportSetDestRect (I-657)

Sets the destination rectangle for a graphics import operation.

GraphicsImportGetDestRect (I-632)

Returns the destination rectangle for an imported image.

GraphicsImportSetFlags (I-658)

Sets the flags for a graphics importer component.

GraphicsImportGetFlags (I-634)

Returns the current flags of a graphics importer component.

Obtaining a Graphics Importer Instance

There are three functions you can call to get a graphics importer instance.

These functions are listed below.

Functions

GetGraphicsImporterForFile (I-414)

Locates and opens a graphics importer component that can be used to draw a specified file.

GetGraphicsImporterForDataRef (I-412)

Locates and opens a graphics importer component that can be used to draw the image from specified data reference.

GetGraphicsImporterForDataRefWithFlags (I-413)

Locates and opens a graphics importer component for a data reference with flags that control the search process.

Getting Image Characteristics

Certain functions are called by applications to obtain information about images.

Format-specific graphics importers always implement

GraphicsImportGetImageDescription (I-638) and may optionally implement the remaining functions listed below.

Functions

GraphicsImportGetNaturalBounds (I–642)

Returns the bounding rectangle of an imported image.

GraphicsImportGetImageDescription (I–638)

Returns image description information for an imported image.

GraphicsImportGetDataOffsetAndSize (I–624)

Returns the offset and size of the compressed image data within an imported image file.

GraphicsImportValidate (I–665)

Validates image data for a data reference to an imported image.

GraphicsImportGetMetaData (I–640)

Extracts user data from an imported image file.

GraphicsImportDoesDrawAllPixels (I–613)

Asks whether the graphics importer expects to draw every pixel.

Setting Drawing Parameters

Certain graphics importer functions allow you to specify various parameters for drawing operations, such as clipping, scaling, graphics mode, and decompression quality. All of these functions are based on corresponding routines in the Image Compression Manager for working with image decompression sequences.

These functions are listed below.

Functions

GraphicsImportSetBoundsRect (I–649)

Defines the rectangle in which to draw an imported image.

GraphicsImportGetBoundsRect (I–619)

Returns the bounding rectangle for drawing an imported image.

GraphicsImportSetMatrix (I–661)

Defines the transformation matrix to use for drawing an imported image.

GraphicsImportGetMatrix (I–639)

Returns the transformation matrix to be used for drawing an imported image.

GraphicsImportSetClip (I-650)

Defines the clipping region for drawing an imported image.

GraphicsImportGetClip (I-620)

Returns the current clipping region for an imported image.

GraphicsImportSetGraphicsMode (I-659)

Sets the graphics transfer mode for an imported image.

GraphicsImportGetGraphicsMode (I-635)

Returns the graphics transfer mode for an imported image.

GraphicsImportSetQuality (I-663)

Sets the image quality value for an imported image.

GraphicsImportGetQuality (I-643)

Returns the image quality value for an imported image.

GraphicsImportSetSourceRect (I-664)

Sets the source rectangle to use for an imported image.

GraphicsImportGetSourceRect (I-645)

Returns the current source rectangle for an imported image.

GraphicsImportSetProgressProc (I-662)

Installs a progress procedure to call while drawing an imported image.

GraphicsImportGetProgressProc (I-643)

Returns the current **progress function** for a graphics import operation.

Drawing Imported Images

Functions are provided to determine the graphics world for an imported graphics image and to draw it.

These functions are listed below.

Functions

GraphicsImportSetGWorld (I-660)

Sets the graphics port and device for drawing an imported image.

GraphicsImportGetGWorld (I-636)

Returns the current graphics port and device for drawing an imported image.

GraphicsImportDraw (I-616)
Draws an imported image.

Saving Image Files

Graphics import components can save data in several formats, including QuickDraw pictures and QuickTime Image files. This capability is only needed by applications that perform file format translation. Applications that only wish to draw the image can use GraphicsImportDraw (I-616).

Graphics import functions for saving image files are listed below.

Functions

GraphicsImportSaveAsPicture (I-647)
Creates a QuickDraw **picture file** for an imported image.

GraphicsImportSaveAsQuickTimeImageFile (I-648)
Creates a QuickTime Image file of an imported image.

GraphicsImportGetAsPicture (I-618)
Creates a QuickDraw picture handle to an imported image.

GraphicsImportExportImageFile (I-616)
Saves an imported image in a foreign file format.

GraphicsImportGetExportImageTypeList (I-632)
Returns information about available export formats for a graphics importer.

GraphicsImportDoExportImageFileDialog (I-614)
Presents a dialog box letting the user save an imported image in a foreign file format.

GraphicsImportGetExportSettingsAsAtomContainer (I-633)
Retrieves settings for image files exported by the graphics importer.

GraphicsImportSetExportSettingsFromAtomContainer (I-657)
Determines settings for the export of imported image files.

Getting MIME Types

Your graphics import component can support MIME types that correspond to graphics formats it supports.

To make a list of these MIME types available to applications or other software, it must implement `GraphicsImportGetMIMETYPEList` (I–641).

Functions

`GraphicsImportGetMIMETYPEList` (I–641)

Returns a list of MIME types supported by the graphics importer component.

Specifying a Graphics Import Data Source

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use the graphics importer component functions described in this section, rather than directly calling a data handler. These functions allow the data source to be a file, a handle, or a QuickTime data reference.

You do not need to call the functions in this section if you use one of the `GetGraphicsImporter` functions. The `GetGraphicsImporter` functions automatically set the graphics importer component's data source. You only need to use these functions if you open the graphics importer component directly.

Functions

`GraphicsImportSetDataFile` (I–651)

Specifies the file that contains imported graphics data.

`GraphicsImportGetDataFile` (I–621)

Returns the file containing the graphics data for an imported image.

`GraphicsImportSetDataHandle` (I–652)

Specifies the handle that references imported graphics data.

`GraphicsImportGetDataHandle` (I–623)

Returns a handle to imported graphics data.

`GraphicsImportSetDataReference` (I–653)

Specifies the data reference for imported graphics data.

`GraphicsImportGetDataReference` (I–626)

Returns a data reference to imported graphics data.

`GraphicsImportSetDataReferenceOffsetAndLimit` (I–654)

Specifies the data reference starting offset and data size limit for an imported image.

`GraphicsImportGetDataReferenceOffsetAndLimit (I-627)`

Returns the data reference starting offset and data size limit for an imported image.

Retrieving Graphics Import Image Data

A function is used by format-specific graphics import components to read data from the data source. It is implemented by the **base graphics importer**.

This function is listed below.

Function

`GraphicsImportReadData (I-645)`

Reads imported image data.

Using Graphics Exporters

Graphics exporter components provide a standard interface for exporting graphics to image files in a variety of formats.

Setting the Input and Output of a Graphics Exporter Instance

One function is used to perform an export operation.

This function is listed below.

Function

`GraphicsExportDoExport (I-554)`

Performs a graphics export operation.

Internal Graphics Export Routines

Certain functions are used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call them.

These functions are listed below.

Functions

GraphicsExportCanTranscode (I-552)

Asks whether the current graphics export operation should be performed by transcoding.

GraphicsExportDoTranscode (I-555)

Performs a graphics export operation by transcoding.

GraphicsExportCanUseCompressor (I-553)

Asks whether to use a compressor in a graphics export operation.

GraphicsExportDoUseCompressor (I-556)

Performs a graphics export operation with compression.

GraphicsExportDoStandaloneExport (I-554)

Performs a standalone graphics export operation.

Finding Out About Graphics Export Image Formats

Certain functions return information about the image format supported by a graphics exporter. Format-specific exporters must implement all three of these functions.

These functions are listed below.

Functions

GraphicsExportGetDefaultFileTypeAndCreator (I-561)

Returns the suggested file type and creator for a graphics export operation.

GraphicsExportGetDefaultFileNameExtension (I-560)

Returns the suggested file name extension for a graphics export operation.

GraphicsExportGetMIMETypeList (I-577)

Returns MIME types and other information about the graphics format in a graphics export operation.

Obtaining Graphics Exporter Settings

Certain functions are used for obtaining graphics exporter settings and displaying a settings dialog box.

These functions are listed below.

Functions

`GraphicsExportRequestSettings` (I-588)

Displays a dialog for the user to configure graphics exporter settings, if applicable.

`GraphicsExportSetSettingsFromAtomContainer` (I-610)

Sets the graphics exporter component's current configuration to match the settings in a passed atom container.

`GraphicsExportGetSettingsAsAtomContainer` (I-583)

Retrieves the current settings from a graphics exporter component.

`GraphicsExportGetSettingsAsText` (I-584)

Retrieves the current settings from the graphics export component in a user-readable format.

Accessing Graphics Exporter Settings

Graphics exporters may implement some or none of the functions listed below.

To determine whether a particular setting is available, use `CallComponentCanDo` (I-58).

Functions

`GraphicsExportSetDontRecompress` (I-592)

Requests that the original compressed data for a graphics export operation not be decompressed and recompressed, but be copied through to the output file.

`GraphicsExportGetDontRecompress` (I-562)

Determines whether the original compressed data for a graphics export operation will not be decompressed and recompressed, but be copied through to the output file.

`GraphicsExportSetInterlaceStyle` (I-603)

Defines the **interlace** style for a graphics export operation.

GraphicsExportGetInterlaceStyle (I–575)

Returns the **interlace** style in a graphics export operation.

GraphicsExportSetMetaData (I–604)

Defines supplemental data for a graphics export operation, such as copyright text.

GraphicsExportGetMetaData (I–576)

Returns the current user data setting in a graphics export operation.

GraphicsExportSetTargetDataSize (I–611)

Defines a desired maximum data size for a graphics export operation and asks for a quality that does not exceed that size.

GraphicsExportGetTargetDataSize (I–585)

Returns the current desired maximum data size for a graphics export operation.

GraphicsExportSetCompressionMethod (I–589)

Defines the compression method to use in a graphics export operation.

GraphicsExportGetCompressionMethod (I–559)

Returns the compression method for a graphics export operation.

GraphicsExportSetCompressionQuality (I–590)

Defines the compression quality for a graphics export operation.

GraphicsExportGetCompressionQuality (I–559)

Returns the compression quality value for a graphics export operation.

GraphicsExportSetResolution (I–609)

Defines the resolution to store in the image file for a graphics export operation.

GraphicsExportGetResolution (I–583)

Determines the resolution of a graphics exporter component.

GraphicsExportSetDepth (I–591)

Defines the depth to use in a graphics export operation.

GraphicsExportGetDepth (I–562)

Returns the current depth setting for a graphics export operation.

GraphicsExportSetColorSyncProfile (I–589)

Sets the ColorSync profile to embed in the image file for a graphics export operation.

GraphicsExportGetColorSyncProfile (I–558)

Gets the current value of the ColorSync profile for a graphics export operation.

Getting and Setting Progress Procs

Functions for getting and setting progress callbacks are always implemented by the **base graphics exporter**.

These functions are listed below.

Functions

GraphicsExportSetProgressProc (I-608)

Installs a **progress function** in a graphics export operation.

GraphicsExportGetProgressProc (I-582)

Returns the current **progress function** for a graphics export operation.

Specifying Sources for Graphics Exporter Input Images

You must specify an input image source before you can call GraphicsExportDoExport (I-554). The source can be a QuickTime graphics importer component instance, a QuickDraw Picture (IV-2331), a graphics world, or a pixel map. Or it can be a piece of compressed data described by an image description. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source.

All of the functions listed below are implemented by the **base graphics exporter**. Format-specific importers should delegate all of them

Functions

GraphicsExportSetInputDataReference (I-593)

Specifies that the source image for a graphics export operation is a compressed image stored in a data reference.

GraphicsExportGetInputDataReference (I-563)

Returns the current input data reference for a graphics export operation.

GraphicsExportSetInputFile (I-594)

Specifies that the source image for a graphics export operation is a compressed image stored in a file.

GraphicsExportGetInputFile (I-565)

Returns the current input file for a graphics export operation.

GraphicsExportSetInputHandle (I–597)

Specifies that the source image for a graphics export operation is a compressed image referenced by a handle.

GraphicsExportGetInputHandle (I–568)

Returns the current input handle for a graphics export operation.

GraphicsExportSetInputPtr (I–602)

Specifies that the source image for a graphics export operation is a compressed image stored at a fixed address in memory.

GraphicsExportGetInputPtr (I–574)

Returns the current input pointer in a graphics export operation.

GraphicsExportSetInputGraphicsImporter (I–595)

Specifies that the source image for a graphics export operation is to be drawn by a graphics importer instance.

GraphicsExportGetInputGraphicsImporter (I–566)

Returns the current input graphics importer instance for a graphics export operation.

GraphicsExportSetInputPicture (I–599)

Specifies that the source image for a graphics export operation is a picture.

GraphicsExportGetInputPicture (I–572)

Returns the current input picture in a graphics export operation.

GraphicsExportSetInputGWorld (I–596)

Specifies that the source image for a graphics export operation is a graphics world.

GraphicsExportGetInputGWorld (I–567)

Returns the current input graphics world for a graphics export operation.

GraphicsExportSetInputPixmap (I–600)

Specifies that the source image for a graphics export operation is a pixmap.

GraphicsExportGetInputPixmap (I–573)

Returns the current input pixmap in a graphics export operation.

Restricting the Range of an Input Image’s Source

Certain functions restrict the range of the input image source for a graphics exporter.

The functions listed below are only applicable when the input is a data reference, file, handle, or pointer.

Functions

`GraphicsExportSetInputOffsetAndLimit` (I-598)

Specifies the portion of an input data reference, file, handle or pointer that a graphics exporter is permitted to read.

`GraphicsExportGetInputOffsetAndLimit` (I-572)

Retrieves the current input offset and limit in a graphics export operation.

Reading Graphics Exporter Input Data

Certain functions read the input data for graphics exporters.

These functions are used by format-specific graphics exporters when transcoding. Applications will not normally need to call them.

Functions

`GraphicsExportMayExporterReadInputData` (I-585)

Asks whether the image source for a graphics export operation is in a form that the exporter can read.

`GraphicsExportGetInputDataSize` (I-564)

Returns the number of bytes of original image data that can be read in a graphics export operation.

`GraphicsExportReadInputData` (I-586)

Reads the original image data in a graphics export operation.

Accessing a Graphics Exporter's Input Image

Certain functions are used by format-specific graphics exporters to access input images.

When doing a standalone export, an exporter will typically call

`GraphicsExportGetInputImageDescription` (I-570) or

`GraphicsExportGetInputImageDimensions` and `GraphicsExportGetInputImageDepth`

(I-569) to determine the image's bounds and depth, and then allocate an offscreen

graphics world and call `GraphicsExportDrawInputImage` (I-557) to draw portions of the image into that world.

Functions

`GraphicsExportGetInputImageDescription` (I-570)

Returns an image description describing the input image in a graphics export operation.

`GraphicsExportGetInputImageDimensions` (I-571)

Returns the dimensions of the input image in a graphics export operation.

`GraphicsExportGetInputImageDepth` (I-569)

Returns the depth of the input image for a graphics export operation.

`GraphicsExportDrawInputImage` (I-557)

Draws a rectangular portion of the input image in a graphics export operation.

Specifying Destinations for Output Images

Certain functions are used to specify destinations for graphics exporter output images.

These functions are listed below.

Functions

`GraphicsExportSetOutputDataReference` (I-604)

Returns the current output data reference for a graphics export operation.

`GraphicsExportGetOutputDataReference` (I-578)

Gets the output data reference handle in a graphics export operation.

`GraphicsExportSetOutputFile` (I-605)

Defines the output file for a graphics export operation.

`GraphicsExportGetOutputFile` (I-578)

Returns the current output file for a graphics export operation.

`GraphicsExportSetOutputHandle` (I-606)

Sets a handle to the output of a graphics export operation.

`GraphicsExportGetOutputHandle` (I-580)

Returns the current output handle for a graphics export operation.

GraphicsExportSetOutputOffsetAndMaxSize (I-608)

Specifies the output starting offset and maximum size limit for a graphics export operation.

GraphicsExportGetOutputOffsetAndMaxSize (I-581)

Returns the output starting offset and maximum size limit for a graphics export operation.

GraphicsExportSetOutputFileTypeAndCreator (I-606)

Sets the file type and creator codes for the output file of a graphics export operation.

GraphicsExportGetOutputFileTypeAndCreator (I-579)

Gets the type and creator codes for the output file in a graphics export operation.

Writing Graphics Exporter Output Data

Certain functions are used by format-specific graphics exporters to write output data.

These functions are listed below.

Functions

GraphicsExportWriteOutputData (I-611)

Writes output image data in a graphics export operation.

GraphicsExportSetOutputMark (I-607)

Seeks to the specified file position in a graphics export operation.

GraphicsExportGetOutputMark (I-580)

Returns the current file position for a graphics export operation.

GraphicsExportReadOutputData (I-587)

Reads output image data in a graphics export operation.

Using Video Outputs

Video output components provide an interface for sending QuickTime video to devices that are not recognized as video displays by a computer's operating system. Any software that presents video data can be a client of a video output component.

Controlling the Video Output Display Mode

Each video output device has a finite number of display modes. Each mode has several characteristics, including width and height of the display, pixel depth, and video refresh rate.

To get a list of the display modes supported by a video output component, call `QTVideoOutputGetDisplayModeList` (II-1326). The list is a QT atom container, and list atoms contain the characteristics of each mode. To specify a display mode to use, call `QTVideoOutputSetDisplayMode` (II-1332). To find out the current display mode, call `QTVideoOutputGetDisplayMode` (II-1325).

Functions

`QTVideoOutputGetDisplayModeList` (II-1326)

Returns a list of the display modes supported by a video output component.

`QTVideoOutputSetDisplayMode` (II-1332)

Specifies the display mode to be used by a video output component.

`QTVideoOutputGetDisplayMode` (II-1325)

Returns the current display mode for a video output component.

Registering the Name of Video Output Software

Several functions help you register the name of your video output component.

After your software has established a connection to a video output component, you can register its name with the instance of that component by calling `QTVideoOutputSetClientName` (II-1332). The name can then be used by `QTVideoOutputGetCurrentClientName` (II-1325) to specify which software has exclusive access to the video output device controlled by the component. To get the

name of the application or other software that is registered with an instance of a video output component, call `QVideoOutputGetClientName` (II–1323).

Functions

`QVideoOutputSetClientName` (II–1332)

Registers the name of an application or other software with an instance of a video output component.

`QVideoOutputGetCurrentClientName` (II–1325)

Returns the name of the software, if any, that has exclusive access to the video hardware controlled by a video output component.

`QVideoOutputGetClientName` (II–1323)

Obtains the name of the application or other software that is registered with an instance of a video output component.

Controlling Video Output

Video output components provide functions for configuring the video display, for starting and stopping video output, and for specifying the graphics world used for the display.

To display a dialog box in which the user can specify video settings, call `QVideoOutputCustomConfigureDisplay` (II–1322). To get a pointer to the graphics world used by the video output component, call `QVideoOutputGetGWorld` (II–1327). Although several applications or other software can connect to a video output component at the same time, only one of them at a time can have access to the video output device controlled by the component. Use `QVideoOutputBegin` (II–1321) to gain exclusive access to the video output device and `QVideoOutputEnd` (II–1322) to relinquish exclusive access when your software has finished using the device. If a video output device can display video both on an external video display and in a window on a computer's desktop, you can use `QVideoOutputSetEchoPort` (II–1333) to specify a window on the desktop in which to display video sent to the device.

Functions

`QVideoOutputCustomConfigureDisplay` (II–1322)

Displays a custom video configuration dialog box, which can include settings that are specific to the video device controlled by the video output component.

`QVideoOutputGetGWorld` (II–1327)

Returns a pointer to the graphics world used by a video output component.

QTVideoOutputGetGWorldParameters (II–1328)

Called by the base video output component as part of its implementation of QTVideoOutputGetGWorld (II–1327).

QTVideoOutputBegin (II–1321)

Obtains exclusive access to the video hardware controlled by a video output component.

QTVideoOutputEnd (II–1322)

Releases access to the video hardware controlled by a video output component.

QTVideoOutputSetEchoPort (II–1333)

Specifies a window on the desktop in which to display video sent to the device.

Finding Components Associated With a Video Output

Video output components provide functions for finding other components associated with them.

To find any sound output components associated with a video output component, call QTVideoOutputGetIndSoundOutput (II–1329). To find a clock component associated with a video output component, call QTVideoOutputGetClock (II–1324).

Functions

QTVideoOutputGetIndSoundOutput (II–1329)

Determines which sound output components are associated with the video output component.

QTVideoOutputGetClock (II–1324)

Returns a pointer to the clock component associated with the video output component.

Saving and Restoring Component Configurations

Video output components provide functions for saving the current configuration of a video output component and later restoring the configuration.

To save the current configuration of a video output component, call QTVideoOutputSaveState (II–1331). To restore a previously saved configuration of a video output component, call QTVideoOutputRestoreState (II–1330).

Functions

QTVideoOutputSaveState (II–1331)

Saves state information for an instance of a video output component.

QTVideoOutputRestoreState (II–1330)

Restores the previously saved state of a video output component.

Using the Standard Sound Dialog

The standard sound dialog component is a QuickTime component with an API for configuring sound settings.

Managing the Sound Dialog

The standard sound dialog component implements three functions from the standard compression component interface.

These functions are listed below.

Functions

SCGetInfo (III–1545)

Retrieves configuration information from the standard dialog component.

SCSetInfo (III–1558)

Modifies the standard dialog component's configuration information.

SCRequestImageSettings (III–1555)

Displays the standard image dialog box to the user and shows default settings you have established.

See Also

See “Using Image-Compression Dialogs” (V–2800).

Sound Dialog Selectors

Several sound constants are in the list of selectors that can be passed to SCGetInfo (III–1545) and SCSetInfo (III–1558). These selectors are only supported by the

standard sound dialog component, not the standard image-compression dialog component.

These selectors are listed below.

Constants

`scSoundSampleRateType`

A pointer to an `UnsignedFixed` value that represents the current sample rate.

`scSoundSampleSizeType`

A pointer to a short integer that represents the current sample size. This value is either 8 or 16.

`scSoundChannelCountType`

A pointer to a short integer that represents the current number of channels. This value is either 1 or 2.

`scSoundCompressionType`

A pointer to a value of type `OSType` that represents the current compression type. This value is either `kRawCodecType` or one of the available sound compression formats.

`scCompressionListType`

A pointer to a handle containing an array of values of type `OSType` that indicate the sound compression formats that may be presented to the user. Pass `NIL` to `SCSetInfo` to reset the list to all available sound compression formats.

`scPreferenceFlagsType`

The only flag that is supported in the preferences is `scUseMovableModal`. All other flags should be set to zero. The preference flags are initialized to `scUseMovableModal` when the standard sound dialog component is instantiated.

`scExtendedProcType`

The only fields supported are `filterProc` and `refcon`. All other fields should be initialized to zero. The filter procedure should only be used to update background windows. It should not be used to intercept user interactions in the dialog box window itself.

`scSettingsStateType`

This selector is supported in the same way as in the standard image-compression component.

Working With Video Effects

QuickTime video effects are implemented as specialized image decompressor components. For more information about implementing image decompressor components, see “Managing Image Compression” (V–2786). To use an effect component in a QuickTime movie, you add an effect track to the movie. The size and duration of the track determines the area of the movie that is affected and how long the effect runs.

High-Level Effects Functions

Certain functions give you high-level pre-packaged access to video effects. These functions are designed to be easy to use and give you access to the most common uses of the QuickTime video effects architecture.

These functions are listed below.

Functions

`QTGetEffectsList` (II–1174)

Returns a QT atom container holding a list of the currently installed effects components.

`QTCreateStandardParameterDialog` (II–1161)

Creates a dialog box that allows the user to choose an effect from the list of effects passed to the function.

`QTIsStandardParameterDialogEvent` (II–1186)

Determines if a Macintosh event is processed by a standard parameter dialog box created by `QTCreateStandardParameterDialog` (II–1161).

`QTDismissStandardParameterDialog` (II–1163)

Closes a standard parameter dialog box that was created using `QTCreateStandardParameterDialog` (II–1161).

`QTStandardParameterDialogDoAction` (II–1313)

Lets you change some of the default behaviors of the standard parameter dialog box.

`QTGetEffectSpeed` (II–1176)

Returns the speed of the effect, expressed in frames per second.

Low-Level Effects Functions

Certain functions provide more complex and comprehensive interfaces to the effects dialog functionality. Using the low-level functions, you can gain more control over the standard parameters dialog box, such as the ability to incorporate user interface elements from the dialog box into your own application-defined dialog box.

These functions are listed below.

Functions

`ImageCodecGetParameterList` (II-717)

Returns a parameter description atom container for a specified effect component instance.

`ImageCodecCreateStandardParameterDialog` (II-692)

Creates a parameters dialog box for a specified effect.

`ImageCodecIsStandardParameterDialogEvent` (II-726)

Processes events related to a standard parameters dialog box created by `ImageCodecCreateStandardParameterDialog` (II-692).

`ImageCodecStandardParameterDialogDoAction` (II-740)

Allows you to control the behavior of a standard parameter dialog box created by `ImageCodecCreateStandardParameterDialog` (II-692).

`ImageCodecDismissStandardParameterDialog` (II-695)

Retrieves values from a standard parameter dialog box created by the low-level `ImageCodecCreateStandardParameterDialog` (II-692) function, then closes the dialog box.

Creating an Effect Sample Description

One function is designed to create a sample description for an effect.

This function is listed below.

Functions

`MakeImageDescriptionForEffect` (II-785)

Returns an `ImageDescription` (IV-2285) structure you can use to help create a sample description for an effect.

Working With the Vector Codec

QuickTime vectors are mathematical descriptions of images that are smaller in size than their equivalent bitmaps and can be scaled without loss of image quality. A vector QT atom container specifies the characteristics of a QuickTime vector. The container contains atoms for paths and can also contain atoms that specify attributes of the paths, such as their color.

Vector Codec Component Functions

Several functions are provided by the vector codec component.

These functions are listed below.

Functions

CurveAddAtomToVectorStream (I-151)

Adds an atom to a **vector data stream**.

CurveAddPathAtomToVectorStream (I-152)

Adds a path to a **vector data stream**.

CurveAddZeroAtomToVectorStream (I-152)

Adds a kCurveEndAtom to a **vector data stream**; this atom marks the end of the vector data stream,

CurveCountPointsInPath (I-153)

Counts the points along either one of a path's contours or all of its contours.

CurveCreateVectorStream (I-154)

Creates a new, empty **vector data stream**.

CurveGetAtomDataFromVectorStream (I-154)

Finds the first atom of a specified type within a **vector data stream** and get its data.

CurveGetLength (I-155)

Calculates the length of one of a path's contours or the sum of the lengths of all of its contours.

CurveGetNearestPathPoint (I-156)

Finds the closest point on a path to a specified point.

CurveGetPathPoint (I-157)

Obtains a point from a path and to find out if the point is on the curve.

CurveInsertPointIntoPath (I-158)

Adds a new point to a path.

CurveLengthToPoint (I-159)

Obtains the point at a specified distance along a curve.

CurveNewPath (I-160)

Creates a new path.

CurvePathPointToLength (I-161)

Obtains the length of a path between specified starting and ending distances that is nearest a point.

CurveSetPathPoint (I-162)

Changes the location of a point in a path.

Working With the Sprite Toolbox

The Sprite Toolbox is a set of data types and functions you can use to add **sprite**-based animation to an application. The Sprite Toolbox composes sprites and their background in an offscreen buffer, invalidates appropriate screen areas as sprite properties change, and transfers the result to the screen or to an alternate destination.

If you're authoring an animation outside of a movie, you use the Sprite Toolbox to create sprite worlds and sprite animations. To create a sprite track within a QuickTime movie, you create media samples used by the sprite media handler, which, in turn, makes use of the Sprite Toolbox.

Working with Sprite Worlds

A **sprite** world is a graphics environment populated by sprites that may exist outside of movies. The Sprite Toolbox provides several functions for creating and manipulating sprite worlds.

These functions are listed below.

Functions

NewSpriteWorld (II–1114)

Creates a new **sprite** world.

DisposeSpriteWorld (I–297)

Disposes of a **sprite** world.

SetSpriteWorldClip (III–1644)

Sets a **sprite** world's clip shape to the specified region.

SetSpriteWorldMatrix (III–1647)

Sets a **sprite** world's matrix to the specified matrix.

SpriteWorldIdle (III–1908)

Allows a **sprite** world to update its invalid areas.

InvalidateSpriteWorld (II–773)

Invalidates a rectangular area of a **sprite** world.

SpriteWorldHitTest (III–1907)

Determines whether any **sprites** are at a specified location in a sprite world.

DisposeAllSprites (I–255)

Disposes of all **sprites** associated with a sprite world.

Creating and Manipulating Sprites

The Sprite Toolbox provides several functions for creating and manipulating **sprites** in sprite worlds.

These functions are listed below.

Functions

NewSprite (II–1113)

Creates a new **sprite** in a specified sprite world.

DisposeSprite (I–296)

Disposes of a **sprite**.

InvalidateSprite (II–772)

Invalidates the portion of a **sprite**'s sprite world that is occupied by a sprite.

SpriteHitTest (III–1885)

Determines whether a location in a **sprite**’s display coordinate system intersects the sprite.

GetSpriteProperty (I–512)

Retrieves the value of a specified **sprite** property.

SetSpriteProperty (III–1641)

Sets the specified property of a **sprite**.

Using the Sprite Media Handler

The **sprite** media handler is a media handler you can use to add a sprite animation track to a QuickTime movie. It provides routines for manipulating the sprites and images in a sprite track. The sprite media handler makes use of the functionality provided by the Sprite Toolbox. If you are using the sprite media handler, you don’t need to use the toolbox API.

Managing Movie Sprites

The **sprite** media handler provides several functions that let you manage sprites in a movie.

These functions are listed below.

Functions

SpriteMediaSetSpriteProperty (III–1904)

Sets the specified property of a **sprite**.

SpriteMediaGetSpriteProperty (III–1896)

Retrieves the value of the specified **sprite** property.

SpriteMediaHitTestAllSprites (III–1897)

Determines whether any **sprites** are at a specified location.

SpriteMediaCountSprites (III–1887)

Retrieves the number of **sprites** that currently exist in a sprite track.

SpriteMediaCountImages (III–1886)

Retrieves the number of images that currently exist in a **sprite** track.

SpriteMediaGetIndImageDescription (III–1891)

Retrieves an image description for a specified image in a **sprite** track.

SpriteMediaGetDisplayedSampleNumber (III–1889)

Retrieves the number of the **sprite** media sample that is currently being displayed.

SpriteMediaGetSpriteName (III–1895)

Returns the name of the **sprite** with the specified ID from the currently displayed sample.

SpriteMediaGetImageName (III–1890)

Returns the name of the image with the specified index from the current **key frame** sample.

SpriteMediaHitTestOneSprite (III–1898)

Performs a hit testing operation on the **sprite** specified by a `spriteID`.

SpriteMediaSpriteIndexToID (III–1906)

Returns the ID of a **sprite** specified by a sprite index.

SpriteMediaSpriteIDToIndex (III–1906)

Converts a **sprite** ID to the corresponding sprite index.

SpriteMediaGetIndImageProperty (III–1891)

Returns a property value for a **sprite** image specified by an index.

Working With Wired Sprites

A wired **sprite** acts like a button, telling other software when the user has clicked on its image or passed the cursor over it. Two functions are specific to sprite tracks that use wired sprites.

These functions are listed below.

Functions

SpriteMediaSetActionVariable (III–1901)

Sets the value of a **sprite** track variable to a specified value.

SpriteMediaGetActionVariable (III–1888)

Returns the value of the **sprite** track variable with the specified ID.

Working With Virtual Reality

QuickTime VR is an extension of the QuickTime technology that allows users to interactively explore and examine photorealistic, three-dimensional virtual worlds. QuickTime VR is presented as a special kind of QuickTime movie, displayed by the VR movie controller. The user navigates in the virtual world that is displayed using standard input devices, such as the mouse and keyboard.

Initializing and Terminating QuickTime VR

The QuickTime VR Manager provides two functions for initializing and terminating its operation. These functions are required for QuickTime VR to run in a Windows environment. They do nothing in the Mac OS environment, but should be included for cross-platform compatibility.

These functions are listed below.

Functions

InitializeQTVR (II-758)

Initializes QuickTime virtual reality.

TerminateQTVR (III-1927)

Terminates QuickTime VR when you have finished using its functions.

Managing QuickTime VR Movie Instances

The QuickTime VR Manager provides functions for obtaining a QTVR track and a new movie instance.

These functions are listed below.

Functions

QTVRGetQTVRTrack (II-1375)

Obtains a QTVR track contained in a QuickTime movie to use in the QTVRGetQTVRInstance (II-1373) call.

QTVRGetQTVRInstance (II-1373)

Obtains an instance of a QuickTime VR movie.

Manipulating Viewing Angles and Zooming

The QuickTime VR Manager provides functions that you can use to manipulate the viewing angles and zooming characteristics of a QuickTime VR movie. Note that the parameters to these functions that specify angles are always interpreted in the current angular unit (degrees or radians) set by a previous call to `QTVRSetAngularUnits` (II–1408). The default angular unit is degrees.

These functions are listed below.

Functions

`QTVRGetPanAngle` (II–1373)

Obtains the pan angle of a QuickTime VR movie.

`QTVRSetPanAngle` (II–1431)

Sets the pan angle of a QuickTime VR movie.

`QTVRGetTiltAngle` (II–1377)

Obtains the tilt angle of a QuickTime VR movie.

`QTVRSetTiltAngle` (II–1434)

Sets the tilt angle of a QuickTime VR movie.

`QTVRGetFieldOfView` (II–1360)

Obtains the vertical field of view of a QuickTime VR movie.

`QTVRSetFieldOfView` (II–1420)

Sets the vertical field of view of a QuickTime VR movie.

`QTVRGetViewCenter` (II–1378)

Obtains the view center of a QuickTime VR movie.

`QTVRSetViewCenter` (II–1437)

Sets the view center of a QuickTime VR movie.

`QTVRNudge` (II–1402)

Turns one step in a particular direction and displays the new view.

`QTVRInteractionNudge` (II–1389)

Translates the image and displays the new view or rotates the object in a particular direction and displays its new appearance.

`QTVRShowDefaultView` (II–1442)

Displays the default view of a QTVR node.

Getting Scene and Node Information

The QuickTime VR Manager provides functions that you can use to get the VR world scene description atom container and to get and set a scene's current node.

These functions are listed below.

Functions

QTVRGetVRWorld (II-1385)

Obtains the VR world atom container for a movie.

QTVRGoToNodeID (II-1386)

Sets the current node of a movie.

QTVRGetCurrentNodeID (II-1359)

Obtains the current node of a movie.

QTVRGetNodeType (II-1372)

Obtains the type of a movie node.

QTVRGetNodeInfo (II-1371)

Obtains the node information atom container that describes a node and all the hot spots in the node.

Managing Hot Spots

The QuickTime VR Manager provides functions that you can use to manage the hot spots of a node.

These functions are listed below.

Functions

QTVRPtToHotSpotID (II-1405)

Obtains the ID of the hot spot, if any, that lies beneath a point.

QTVRGetHotSpotType (II-1364)

Obtains the type of a QuickTime VR hot spot.

QTVRTriggerHotSpot (II-1443)

Triggers a QTVR hot spot.

QTVREnableHotSpot (II-1340)

Enables or disables one or more QTVR hot spots.

QTVRSetMouseOverHotSpotProc (II-1429)

Installs or removes a mouse over hot spot procedure.

QTVRGetVisibleHotSpots (II-1384)

Obtains a list of the currently visible hot spots in a QuickTime VR movie.

QTVRGetHotSpotRegion (II-1363)

Obtains the region occupied by a hot spot.

Handling Events

The QuickTime VR Manager provides a number of functions that you can use to handle user actions such as moving the mouse or clicking the mouse button. Normally, QuickTime VR handles all user interaction internally if your application calls `MCIsPlayerEvent` (II-819) in its main event loop. For code that does not have a main event loop, you might need to override QuickTime VR's mouse event handling.

These functions are listed below.

Functions

QTVRGetMouseOverTracking (II-1370)

Obtains the current state of mouse-over tracking.

QTVRSetMouseOverTracking (II-1431)

Sets the state of mouse-over tracking.

QTVRMouseEnter (II-1392)

Handles the user's moving the cursor into a QuickTime VR movie for which mouse-over tracking is disabled.

QTVRMouseWithin (II-1401)

Handles the user's leaving the cursor in a QuickTime VR movie for which mouse-over tracking is disabled.

QTVRMouseLeave (II-1394)

Handles the user's moving the cursor out of a QuickTime VR movie for which mouse-over tracking is disabled.

QTVRGetMouseDownTracking (II-1370)

Obtains the current state of mouse-down tracking.

QTVRSetMouseDownTracking (II–1429)

Sets the state of mouse-down tracking.

QTVRMouseDown (II–1391)

Handles the user’s clicking the mouse button when the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

QTVRMouseDown (II–1395)

Handles the user’s holding down the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

QTVRMouseUp (II–1398)

Handles the user’s releasing the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

QTVRMouseDownExtended (II–1396)

Handles the user’s holding down the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

QTVRMouseUpExtended (II–1399)

Handles the user’s releasing the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

Intercepting QuickTime VR Manager Routines

The QuickTime VR Manager provides functions that you can use to install and call intercept procedures. Your intercept procedure can provide either alternative or additional functionality to that provided by the intercepted procedure.

These functions are listed below.

Functions

QTVRInstallInterceptProc (II–1387)

Installs or removes an intercept procedure for a QuickTime VR Manager function.

QTVRCallInterceptedProc (II–1336)

Calls an intercepted QuickTime VR function from within an intercept procedure.

Managing Object Nodes

The QuickTime VR Manager provides functions that you can use to manage object nodes in VR object movies.

These functions are listed below.

Functions

QTVRGetCurrentMouseMode (II–1358)

Obtains the current mouse control modes.

QTVRGetFrameRate (II–1362)

Obtains the current frame rate of an object node.

QTVRSetFrameRate (II–1421)

Sets the frame rate of an object node.

QTVRGetViewRate (II–1381)

Obtains the current view rate of an object node.

QTVRSetViewRate (II–1439)

Sets the view rate of a QTVR object node.

QTVRGetCurrentViewDuration (II–1360)

Obtains the duration of the current view of an object node.

QTVRGetViewCurrentTime (II–1379)

Obtains the current time in the current view.

QTVRSetViewCurrentTime (II–1438)

Sets the time in the current QTVR view.

QTVRGetViewStateCount (II–1383)

Obtains the number of view states of an object node.

QTVRGetViewState (II–1382)

Obtains the value of a view state.

QTVRSetViewState (II–1440)

Sets the value of a QTVR view state.

QTVRGetAnimationSetting (II–1344)

Obtains the current state of an animation setting for an object node.

QTVRSetAnimationSetting (II–1409)

Sets the state of an animation setting for an object node.

QTVRGetControlSetting (II–1355)

Obtains the current state of a control setting for an object node.

QTVRSetControlSetting (II–1416)

Sets the state of a control setting for a QTVR object node.

QTVRGetFrameAnimation (II–1361)

Obtains the current state of frame animation for an object node.

QTVREnableFrameAnimation (II–1339)

Enables or disables frame animation for an object node.

QTVRGetViewAnimation (II–1377)

Obtains the current state of view animation for an object node.

QTVREnableViewAnimation (II–1342)

Enables or disables view animation for an object node.

Managing Imaging Characteristics

The QuickTime VR Manager provides functions that you can use to get and set various imaging characteristics of a VR movie.

These functions are listed below.

Functions

QTVRGetVisible (II–1384)

Obtains a movie's visibility state.

QTVRSetVisible (II–1441)

Sets a VR movie's visibility state.

QTVRGetImagingProperty (II–1365)

Obtains the current value of an imaging property of a movie.

QTVRSetImagingProperty (II–1422)

Sets the value of an imaging property of a movie.

QTVRUpdate (II–1445)

Forces an immediate update of a QuickTime VR movie image.

QTVRBeginUpdateStream (II–1335)

Begins a stream of immediate updates to a QuickTime VR movie.

QTVREndUpdateStream (II-1343)

Ends a stream of immediate updates to a QuickTime VR movie.

QTVRSetTransitionProperty (II-1435)

Sets the value of a transition property.

QTVREnableTransition (II-1341)

Enables or disables a transition effect.

Converting Angles and Points

The QuickTime VR Manager provides several functions that you can use to convert mathematical entities and perform other utility operations.

These functions are listed below.

Functions

QTVRGetAngularUnits (II-1344)

Obtains the type of unit currently used when specifying angles.

QTVRSetAngularUnits (II-1408)

Sets the type of unit used when specifying QTVR angles.

QTVRPtToAngles (II-1404)

Obtains the pan and tilt angles of a point.

QTVRCoordToAngles (II-1338)

Get the pan and tilt angles of a floating-point coordinate in a panorama.

QTVRAnglesToCoord (II-1334)

Obtains a floating-point coordinate determined by a pair of pan and tilt angles.

QTVRPanToColumn (II-1403)

Obtains the column number in the object image array that corresponds to a pan angle.

QTVRColumnToPan (II-1337)

Get the pan angle that corresponds to a column number in the object image array.

QTVRTiltToRow (II-1443)

Obtains the row number in the QTVR object image array that corresponds to a tilt angle.

QTVRRowToTilt (II–1407)

Obtains the tilt angle that corresponds to a row number in a QTVR object image array.

QTVRWrapAndConstrain (II–1446)

Preflights a change in the viewing or control characteristics of a QTVR object or panoramic node.

Managing QuickTime VR Movie Interactions

The QuickTime VR Manager provides functions that you can use to augment the normal user interaction handling provided by a QuickTime movie controller.

These functions are listed below.

Functions

QTVRSetEnteringNodeProc (II–1419)

Installs or removes a node-entering procedure.

QTVRSetLeavingNodeProc (II–1428)

Installs or removes a node-leaving procedure.

QTVRGetInteractionProperty (II–1368)

Obtains the value of an interaction property.

QTVRSetInteractionProperty (II–1425)

Sets the value of an interaction property.

QTVRReplaceCursor (II–1407)

Replaces any of the standard QuickTime VR cursors with your own custom cursor.

Determining Viewing Limits and Constraints

The QuickTime VR Manager provides functions that you can use to get the physical viewing limits of a movie and to get and set a movie's constraints.

These functions are listed below.

Functions

QTVRGetViewingLimits (II–1379)

Obtains the current viewing limits of a QuickTime VR movie.

QTVRGetConstraintStatus (II–1353)

Obtains the set of constraints active for the current view.

QTVRGetConstraints (II–1352)

Obtains the current constraints of a QuickTime VR movie.

QTVRSetConstraints (II–1415)

Sets the constraints of a VR movie.

Managing VR Memory

The QuickTime VR Manager provides functions that you can use to help manage the memory used by QuickTime VR.

These functions are listed below.

Functions

QTVRGetAvailableResolutions (II–1346)

Obtains the image resolutions present in the current node.

QTVRGetBackBufferMemInfo (II–1347)

Obtains information about the internal back buffer that QuickTime VR maintains for caching panoramic images.

QTVRGetBackBufferSettings (II–1350)

Obtains information about the resolution, pixel format, and size of the back buffer maintained internally by QuickTime VR for caching a panoramic image in a particular pixel format.

QTVRSetBackBufferPrefs (II–1412)

Sets the resolution, pixel format, and size of the back buffer maintained internally by QuickTime VR for caching a panoramic image in a particular pixel format.

Accessing Image Buffers

The QuickTime VR Manager provides functions that you can use to access the two internal buffers used by QuickTime VR when it's imaging a panorama.

These functions are listed below.

Functions

QTVRSetPrescreenImagingCompleteProc (II-1432)

Installs or removes a prescreen buffer imaging completion procedure.

QTVRSetBackBufferImagingProc (II-1411)

Installs or removes a QTVR back buffer imaging procedure.

QTVRRefreshBackBuffer (II-1406)

Refreshes the QTVR back buffer.

Programming Music

The QuickTime music architecture (QTMA) allows QuickTime movies, applications, and other software to play individual musical notes, sequences of notes, and a broad range of sounds from a variety of instruments and synthesizers. With QTMA, you can also import Standard MIDI files and convert them into QuickTime movies for easy playback.

Using the Tune Player

The QTMA tune player provides functions for setting, queueing, and manipulating music sequences. It also provides certain utility functions.

These functions are listed below.

Functions

TuneSetHeader (III-1967)

Prepares the tune player to accept subsequent music event sequences by defining one or more parts to be used by sequence Note events.

`TuneSetHeaderWithSize` (III–1968)

Similar to `TuneSetHeader` (III–1967) but lets you specify the header length.

`TuneSetNoteChannels` (III–1969)

Assigns note channels to a tune player.

`TuneQueue` (III–1964)

Places a sequence of music events into a queue to be played.

`TuneStop` (III–1975)

Stops a currently playing sequence.

`TuneGetVolume` (III–1963)

Returns the volume associated with an entire tune sequence.

`TuneSetVolume` (III–1974)

Sets the volume for an entire sequence.

`TuneSetSoundLocalization` (III–1972)

Passes sound localization data to a tune player.

`TuneGetTimeBase` (III–1961)

Returns the time base of the tune player.

`TuneGetTimeScale` (III–1962)

Returns the current time scale for a specified tune player instance.

`TuneSetTimeScale` (III–1973)

Sets the time scale used by the specified tune player instance.

`TuneGetPartMix` (III–1960)

Gets volume, balance, and mixing settings for a specified part of a tune.

`TuneSetPartMix` (III–1970)

Sets volume, balance, and mixing settings for a specified part of a tune.

`TuneInstant` (III–1963)

Plays a particular sequence of events active at a specified position.

`TunePreroll` (III–1964)

Prepares to play a tune player sequence data by attempting to reserve note channels for each part in the sequence.

`TuneUnroll` (III–1976)

Releases any note channel **resources** that may have been locked down by previous calls to `TunePreroll` (III–1964) for this tune player.

`TuneGetIndexedNoteChannel` (III–1958)

Determines how many parts a tune is playing and which instrument is assigned to those parts.

`TuneGetStatus` (III–1961)

Returns an initialized structure describing the state of the tune player instance.

`TuneSetPartTranspose` (III–1971)

Modifies the pitch and volume of every note of a tune.

`TuneGetNoteAllocator` (III–1959)

Returns the instance of the note allocator that the tune player is using.

`TuneSetSofter` (III–1972)

Adjusts the volume a tune is played at to the softer volume produced by QuickTime 2.1.

`TuneSetBalance` (III–1966)

Modifies the pan controller setting for a tune player.

`TuneTask` (III–1975)

Lets a tune player to perform tasks it must perform at foreground task time.

Allocating and Using Note Channels

Several QTMA functions create, manipulate, and get information about note channels.

These functions are listed below.

Functions

`NANewNoteChannel` (II–1030)

Requests a new note channel with the qualities described in a `NoteRequest` (IV–2323) structure.

`NANewNoteChannelFromAtomicInstrument` (II–1031)

Requests a new note channel for an atomic instrument.

`NADisposeNoteChannel` (II–1020)

Deletes a specified note channel.

`NAGetNoteChannelInfo` (II–1026)

Returns the index of the music component for the allocated channel and its part number on that music component.

NAGetIndNoteChannel (II-1023)

Returns the number of note channels handled by the specified note allocator instance.

NAUseDefaultMIDIInput (II-1052)

Defines an entry point to service external MIDI device events.

NALoseDefaultMIDIInput (II-1029)

Removes the external default MIDI service procedure call, if previously defined by **NAUseDefaultMIDIInput (II-1052)**.

NAPrerollNoteChannel (II-1037)

Attempts to reallocate the note channel if it was invalid previously.

NAUnrollNoteChannel (II-1051)

Marks a note channel as available to be stolen.

NAResetNoteChannel (II-1039)

Turns off all currently “on” notes on the note channel and resets all controllers to their default values.

NASetNoteChannelVolume (II-1047)

Sets the volume on the specified note channel.

NASetNoteChannelBalance (II-1046)

Modifies the pan controller setting for a note channel.

NASetNoteChannelSoundLocalization (II-1047)

Passes sound localization data to a note channel.

NAPlayNote (II-1036)

Plays a note with a specified pitch and velocity on the specified note channel.

NAGetController (II-1022)

Retrieves the controller settings for a note channel.

NASetController (II-1042)

Changes the controller setting on a note channel to a specified value.

NAGetKnob (II-1024)

Obtains the value of a knob for a given note channel.

NASetKnob (II-1045)

Sets a note channel knob to a particular value.

`NAFindNoteChannelTone` (II–1021)

Locates the instrument that best fits a requested tone description for a specific channel.

`NASetInstrumentNumber` (II–1044)

Initializes a synthesizer part with the specified instrument.

`NASetInstrumentNumberInterruptSafe` (II–1044)

Initializes a synthesizer part with the specified instrument during interrupt time.

`NASetAtomicInstrument` (II–1041)

Initializes a synthesizer part with an atomic instrument.

`NASendMIDI` (II–1040)

Sends a MIDI music packet to a synthesizer that contains a specific note channel.

`NAGetNoteRequest` (II–1027)

Retrieves the `NoteRequest` (IV–2323) structure that was passed to a note channel.

Note Allocator Interface Tools

Several QTMA functions provide a user interface for instrument selection and presentation of copyright information.

These functions are listed below.

Functions

`NAPickInstrument` (II–1035)

Presents a user interface for picking an instrument.

`NAPickEditInstrument` (II–1033)

Presents a user interface for changing the instrument in a live note channel or modifying an atomic instrument.

`NASTuffToneDescription` (II–1048)

Initializes a tone description structure with the details of a General MIDI note channel.

`NAPickArrangement` (II–1032)

Displays a dialog box to allow instrument selection.

NACopyrightDialog (II–1019)

Displays a copyright dialog box with information specific to a music device.

Note Allocator Configuration and Utilities

Several QTMA functions create and maintain a database of music components, save configuration information in the QuickTime Preferences file, establish connections to external MIDI devices, and allow the note allocator to perform necessary tasks at task foreground time.

These functions are listed below.

Functions

NARegisterMusicDevice (II–1038)

Registers a music component with the note allocator.

NAUnregisterMusicDevice (II–1051)

Removes a previously registered music component from the note allocator.

NAGetRegisteredMusicDevice (II–1028)

Returns details about music components registered to the specified note allocator instance.

NAGetDefaultMIDIInput (II–1023)

Obtains external MIDI connection information.

NASetDefaultMIDIInput (II–1043)

Initializes an external MIDI device used to receive an external note input.

NAGetMIDIPorts (II–1025)

The MIDI input and output ports available to a note allocator.

NASaveMusicConfiguration (II–1039)

Saves the current list of registered devices to a file.

NATask (II–1049)

Called periodically to allow the note allocator to perform tasks in foreground task time.

Managing Synthesizers

Several QTMA functions obtain specific information about a synthesizer and obtain a best instrument fit for a requested tone from the available instruments within the synthesizer; play a note with a specified pitch, volume, and duration; get and set a particular synthesizer knob; obtain synthesizer knob information; and get and set external MIDI procedure name entry points.

These functions are listed below.

Functions

`MusicGetDescription` (II-986)

Returns a structure describing the synthesizer controlled by the music component device.

`MusicFindTone` (II-981)

Returns the number of the best-matching instrument provided by a specified music component.

`MusicPlayNote` (II-1004)

Plays a note on a specified part at a specified pitch and velocity.

`MusicGetKnob` (II-994)

Returns the value of the specified global synthesizer knob.

`MusicSetKnob` (II-1007)

Modifies the value of the specified global synthesizer knob.

`MusicGetKnobDescription` (II-995)

Returns a pointer to an initialized knob description structure describing a global synthesizer knob.

`MusicGetInstrumentKnobDescription` (II-992)

Obtains the description of an instrument knob.

`MusicGetDrumKnobDescription` (II-988)

Returns a description of a drum kit knob.

`MusicGetKnobSettingStrings` (II-996)

Returns a list of knob setting names known by the specified music component.

`MusicSetMIDIProc` (II-1009)

Informs the music component what procedure to call when it needs to send MIDI data.

`MusicGetMIDIProc` (II–998)

Returns a pointer to the procedure a music component is using to process external MIDI notes.

`MusicGetMIDIPorts` (II–997)

Returns the number of input and output ports a MIDI device has.

`MusicSendMIDI` (II–1006)

Sends a MIDI packet to a specified port.

`MusicGetDeviceConnection` (II–987)

Determines how many hardware synthesizers are available to a music component and gets the IDs for those devices.

`MusicUseDeviceConnection` (II–1019)

Tells a music component which hardware synthesizer to talk to.

Managing Instruments and Parts

The QTMA provides functions to initialize a part with an instrument, store instruments, list available instruments, manipulate parts, and get information about parts.

These functions are listed below.

Functions

`MusicGetPartInstrumentNumber` (II–1002)

Returns the instrument number currently assigned to a part.

`MusicSetPartInstrumentNumber` (II–1013)

Superseded by `MusicSetPartInstrumentNumberInterruptSafe` (II–1013).

`MusicSetPartInstrumentNumberInterruptSafe` (II–1013)

Initializes a part with a particular instrument.

`MusicGetPartAtomicInstrument` (II–1000)

Returns the atomic instrument currently in a part.

`MusicSetPartAtomicInstrument` (II–1011)

Initializes a part with an atomic instrument.

`MusicStorePartInstrument` (II–1017)

Puts whatever instrument is on the specified part into the synthesizer's instrument store.

`MusicGetInstrumentAboutInfo` (II-990)

Obtains the information about an instrument that appears in its About box.

`MusicGetInstrumentInfo` (II-991)

Obtains a list of instruments supported by a synthesizer.

`MusicGetPart` (II-999)

Returns the MIDI channel and maximum polyphony for a particular part.

`MusicSetPart` (II-1010)

Sets the MIDI channel and maximum polyphony for a specified part.

`MusicGetPartName` (II-1003)

Returns the string name of a part.

`MusicSetPartName` (II-1015)

Changes the name of an instrument in a specified part.

`MusicGetPartKnob` (II-1002)

Retrieves the current value of a knob for a part.

`MusicSetPartKnob` (II-1014)

Sets a knob for a specified part.

`MusicResetPart` (II-1006)

Silences all sounds on a specified part and resets all controllers on that part to their default values.

`MusicGetPartController` (II-1001)

Returns the value of a specified controller on a specified part.

`MusicSetPartController` (II-1012)

Initializes the value of a specified controller on a specified part.

`MusicSetPartSoundLocalization` (II-1015)

Passes sound localization data to a specified synthesizer part.

Miscellaneous Music Component Functions

You can use several QTMA functions to get and modify the master tuning of the synthesizer, to play off line, and to allow the music component to perform tasks it must perform at foreground task time.

These functions are listed below.

Functions

`MusicGetMasterTune` (II–997)

Returns the synthesizer's master tuning as a fixed-point value in semitones.

`MusicSetMasterTune` (II–1008)

Alters a synthesizer's master tuning.

`MusicStartOffline` (II–1016)

Informs the QuickTime music synthesizer that the music will not be played through the speakers.

`MusicSetOfflineTimeTo` (II–1009)

Advances the synthesizer clock when the synthesizer is not running in real time.

`MusicTask` (II–1018)

Allows a music component to perform tasks it must perform at foreground task time.

Instrument Component Functions

You can use several QTMA functions that are implemented by instrument components.

These functions are listed below.

Functions

`InstrumentGetInfo` (II–766)

Returns information about all the atomic instruments supported by an instrument component.

`InstrumentGetInst` (II–767)

Returns an atomic instrument.

`InstrumentInitialize` (II–769)

Tells the base class instrument component how to interpret the instrument component **resources**.

`InstrumentOpenComponentResFile` (II–770)

Opens a **resource** file containing instruments in the instrument component and makes it the current resource file.

InstrumentCloseComponentResFile (II-765)

Closes a **resource** file.

InstrumentGetComponentRefCon (II-766)

Obtains the reference constant for an instrument component.

MIDI Component Functions

You can use several QTMA functions that are implemented by MIDI components. These functions are MIDI device drivers; they are called by the note allocator MIDI routines.

These functions are listed below.

Functions

QTMIDIGetMIDIPorts (II-1188)

Returns two lists of MIDI ports supported by the specified MIDI component: a list of ports that can receive MIDI input and a list of ports that can send MIDI output.

QTMIDISendMIDI (II-1189)

Sends MIDI data to a MIDI port.

QTMIDIUseReceivePort (II-1189)

Allocates a MIDI port for input or to release the port.

QTMIDIUseSendPort (II-1190)

Allocates a MIDI port for output or to release the port.

Importing MIDI Files

You can use QTMA functions to control the importation of MIDI files.

These functions are listed below.

Functions

MIDIImportGetSettings (II-917)

Obtains settings that control the importation of MIDI files.

MIDIImportSetSettings (II-917)

Define settings that control the importation of MIDI files.

Managing the Generic Music Component

One QTMA functions lets you tell the generic music component what services your music component requires.

This function is listed below.

Functions

`MusicGenericConfigure` (II–982)

Informs the generic music component what services your music component requires and points to any **resources** that are necessary.

Calling Generic Music Component Clients

Several functions are implemented by client music components of the generic music component. They are called by the generic music component, which make calls that are necessary for responding to function calls made directly by applications.

These functions are listed below.

Functions

`MusicDerivedSetKnob` (II–976)

Called when any of the synthesizer's knobs are altered.

`MusicDerivedSetPart` (II–979)

Sets the polyphony for the part specified in the `GCPart` (IV–2263) structure.

`MusicDerivedSetInstrument` (II–976)

The complete instrument defined by the `Part` structure to the synthesizer.

`MusicDerivedSetMIDI` (II–978)

Sets the MIDI channel and other MIDI settings for MIDI output only.

Index of Data Types

AbsoluteTime	“Timing and Callback Types” (IV-2640)
ActionsProcPtr	callback ActionsProc (III-2075)
ActionsUPP	“Universal Procedure Pointers” (IV-2641)
ActivateYDProcPtr	callback ActivateYDProc (III-2075)
ActivateYDUPP	“Universal Procedure Pointers” (IV-2641)
ActivationOrderListPtr	“Sound Types” (IV-2633)
AliasHandle	“File System Types” (IV-2624)
AliasInfoType	“File System Types” (IV-2624)
AliasPtr	“File System Types” (IV-2624)
AliasRecord	struct AliasRecord (IV-2159)
'AllF'	atom 'AllF' (IV-2511)
AreaOfInterest	“Virtual Reality Types” (IV-2647)
ARGBColor	struct ARGBColor (IV-2159)
AtomicInstrument	“Music Types” (IV-2630)
AtomicInstrumentPtr	“Music Types” (IV-2630)
AudioEndianAtomPtr	“Sound Types” (IV-2633)
AudioFormatAtomPtr	“Sound Types” (IV-2633)
AudioInfo	struct AudioInfo (IV-2160)
AudioInfoPtr	“Sound Types” (IV-2633)
AudioSelection	struct AudioSelection (IV-2161)
AudioSelectionPtr	“Sound Types” (IV-2633)
AudioTerminatorAtomPtr	“Sound Types” (IV-2633)
BackBufferImagingProcPtr	“Virtual Reality Types” (IV-2647)
BackBufferImagingUPP	“Virtual Reality Types” (IV-2647)
BandwidthManagementPrefsHandle	“Movie Types” (IV-2628)
BandwidthManagementPrefsPtr	“Movie Types” (IV-2628)
BandwidthManagementPrefsRecord	struct BandwidthManagementPrefsRecord (IV-2161)
BigEndianFixed	struct BigEndianFixed (IV-2161)
BigEndianLong	struct BigEndianLong (IV-2162)
BigEndianOSType	struct BigEndianOSType (IV-2162)
BigEndianShort	struct BigEndianShort (IV-2162)
BigEndianUnsignedFixed	struct BigEndianUnsignedFixed (IV-2163)
BigEndianUnsignedLong	struct BigEndianUnsignedLong (IV-2163)
BigEndianUnsignedShort	struct BigEndianUnsignedShort (IV-2164)
BitMap	struct BitMap (IV-2164)
BitMapHandle	“Graphics Types” (IV-2624)
BitMapPtr	“Graphics Types” (IV-2624)
BooleanRangeRecord	struct BooleanRangeRecord (IV-2165)
Byte	“Numeric Types” (IV-2631)

ByteCount	"Memory Types" (IV-2627)
ByteOffset	"Memory Types" (IV-2627)
BytePtr	"Memory Types" (IV-2627)
CallBackRecord	struct CallBackRecord (IV-2165)
CaretHookUPP	"Universal Procedure Pointers" (IV-2641)
CDSequenceDataSource	struct CDSequenceDataSource (IV-2165)
CDSequenceDataSourcePtr	"Codec Types" (IV-2619)
CDSequenceDataSourceQueueEntry	struct CDSequenceDataSourceQueueEntry (IV-2167)
CDSequenceDataSourceQueueEntryPtr	"Codec Types" (IV-2619)
CGrafPort	struct CGrafPort (IV-2168)
CGrafPtr	"Graphics Types" (IV-2624)
'chap'	atom 'chap' (IV-2512)
CharParameter	"Graphics Types" (IV-2624)
CharsHandle	"String Types" (IV-2637)
CharsPtr	"String Types" (IV-2637)
CleanApertureImageDescriptionExtension	struct CleanApertureImageDescriptionExtension (IV-2174)
'clip'	atom 'clip' (IV-2513)
'cmov'	atom 'cmov' (IV-2514)
CmpSoundHeader	struct CmpSoundHeader (IV-2176)
CmpSoundHeaderPtr	"Sound Types" (IV-2633)
'cmvd'	atom 'cmvd' (IV-2514)
'co64'	atom 'co64' (IV-2515)
CodecCapabilities	struct CodecCapabilities (IV-2179)
CodecComponent	"Codec Types" (IV-2619)
CodecCompressParams	struct CodecCompressParams (IV-2184)
CodecDecompressParams	struct CodecDecompressParams (IV-2190)
CodecFlags	"Codec Types" (IV-2619)
CodecInfo	struct CodecInfo (IV-2202)
CodecNameSpec	struct CodecNameSpec (IV-2208)
CodecNameSpecList	struct CodecNameSpecList (IV-2208)
CodecNameSpecListPtr	"Codec Types" (IV-2619)
CodecQ	"Codec Types" (IV-2619)
CodecType	"Codec Types" (IV-2619)
ColorComplementUPP	"Universal Procedure Pointers" (IV-2641)
ColorSearchUPP	"Universal Procedure Pointers" (IV-2641)
ColorSpec	struct ColorSpec (IV-2209)
ColorSpecPtr	"Graphics Types" (IV-2624)
ColorTable	struct ColorTable (IV-2210)
Component	"Component Types" (IV-2621)
ComponentAliasResource	struct ComponentAliasResource (IV-2211)
ComponentDescription	struct ComponentDescription (IV-2212)
ComponentFunctionUPP	"Component Types" (IV-2621)
ComponentInstance	"Component Types" (IV-2621)
ComponentInstanceRecord	struct ComponentInstanceRecord (IV-2215)
ComponentMPWorkFunctionHeaderRecord	struct ComponentMPWorkFunctionHeaderRecord (IV-2215)

ComponentMPWorkFunctionHeaderRecordPtr	"System and Toolbox Types" (IV-2638)
ComponentMPWorkFunctionProcPtr	callback ComponentMPWorkFunctionProc (III-2077)
ComponentMPWorkFunctionUPP	"Universal Procedure Pointers" (IV-2641)
ComponentParameters	struct ComponentParameters (IV-2216)
ComponentPlatformInfo	struct ComponentPlatformInfo (IV-2218)
ComponentPlatformInfoArray	struct ComponentPlatformInfoArray (IV-2219)
ComponentRecord	struct ComponentRecord (IV-2219)
ComponentResource	struct ComponentResource (IV-2219)
ComponentResourceExtension	struct ComponentResourceExtension (IV-2221)
ComponentResourceHandle	"Component Types" (IV-2621)
ComponentResourcePtr	"Component Types" (IV-2621)
ComponentResult	"Component Types" (IV-2621)
ComponentRoutineProcPtr	callback ComponentRoutineProc (III-2077)
ComponentRoutineUPP	"Universal Procedure Pointers" (IV-2641)
CompressionInfo	struct CompressionInfo (IV-2222)
CompressionInfoHandle	"Sound Types" (IV-2633)
CompressionInfoPtr	"Sound Types" (IV-2633)
CompressorComponent	"Codec Types" (IV-2619)
CompTimeValue	"Timing and Callback Types" (IV-2640)
ConnectionSpeedPrefsHandle	"Movie Types" (IV-2628)
ConnectionSpeedPrefsPtr	"Movie Types" (IV-2628)
ConnectionSpeedPrefsRecord	struct ConnectionSpeedPrefsRecord (IV-2223)
ConstComponentListPtr	"Component Types" (IV-2621)
ConstFSSpecPtr	"File System Types" (IV-2624)
ConstLogicalAddress	"Memory Types" (IV-2627)
ConstSFTYPEListPtr	"Sound Types" (IV-2633)
ConstStr15Param	"String Types" (IV-2637)
ConstStr255Param	"String Types" (IV-2637)
ConstStr27Param	"String Types" (IV-2637)
ConstStr31Param	"String Types" (IV-2637)
ConstStr32Param	"String Types" (IV-2637)
ConstStr63Param	"String Types" (IV-2637)
ConstStrFileNameParam	"String Types" (IV-2637)
ConstStringPtr	"String Types" (IV-2637)
ControlBehaviors	struct ControlBehaviors (IV-2224)
ConversionBlock	struct ConversionBlock (IV-2225)
ConversionBlockPtr	"Sound Types" (IV-2633)
'@cpy'	atom '@cpy' (IV-2515)
'@day'	atom '@day' (IV-2516)
'@dir'	atom '@dir' (IV-2517)
'@ed1'	atom '@ed1' (IV-2517)
'@fmt'	atom '@fmt' (IV-2518)
'@inf'	atom '@inf' (IV-2519)
'@prd'	atom '@prd' (IV-2519)
'@prf'	atom '@prf' (IV-2520)
'@req'	atom '@req' (IV-2521)
'@src'	atom '@src' (IV-2521)

'@wrt'	atom '@wrt' (IV-2522)
CProcHnd1	"Graphics Types" (IV-2624)
CProcPtr	"Graphics Types" (IV-2624)
CProcRec	struct CProcRec (IV-2225)
CQDProcs	struct CQDProcs (IV-2226)
CQDProcsPtr	"Graphics Types" (IV-2624)
createMovieFileFlagsEnum	"Movie Types" (IV-2628)
'crgn'	atom 'crgn' (IV-2523)
'crsr'	atom 'crsr' (IV-2524)
'cspd'	atom 'cspd' (IV-2524)
CSpecArray	"Graphics Types" (IV-2624)
'ctab'	atom 'ctab' (IV-2525)
CTabHandle	"Graphics Types" (IV-2624)
CTabPtr	"Graphics Types" (IV-2624)
'cufa'	atom 'cufa' (IV-2525)
'CURS'	atom 'CURS' (IV-2526)
CursorRecord	"Virtual Reality Types" (IV-2647)
'cuvw'	atom 'cuvw' (IV-2526)
CWindowPtr	"Graphics Types" (IV-2624)
'dasz'	atom 'dasz' (IV-2527)
DataCodecComponent	"Component Types" (IV-2621)
DataCompressorComponent	"Component Types" (IV-2621)
DataDecompressorComponent	"Component Types" (IV-2621)
dataHandleHandle	"Media Handling Types" (IV-2626)
dataHandlePtr	"Media Handling Types" (IV-2626)
DataHandler	"Data Handling Types" (IV-2622)
DataHandlerComponent	"Data Handling Types" (IV-2622)
DataHChokeAtomRecord	struct DataHChokeAtomRecord (IV-2228)
DataHCompletionProcPtr	callback DataHCompletionProc (III-2078)
DataHCompletionUPP	"Universal Procedure Pointers" (IV-2641)
DataHSchedulePtr	"Data Handling Types" (IV-2622)
DataHScheduleRecord	struct DataHScheduleRecord (IV-2229)
DataHVolumeList	"Data Handling Types" (IV-2622)
DataHVolumeListPtr	"Data Handling Types" (IV-2622)
DataHVolumeListRecord	struct DataHVolumeListRecord (IV-2230)
DataRateParams	struct DataRateParams (IV-2231)
DataRateParamsPtr	"Codec Types" (IV-2619)
DataRefAtom	"Atom Types" (IV-2619)
dataRefAttributesFlags	"Movie Types" (IV-2628)
DataReferencePtr	"Media Handling Types" (IV-2626)
DataReferenceRecord	struct DataReferenceRecord (IV-2233)
'dcom'	atom 'dcom' (IV-2527)
DecompressorComponent	"Codec Types" (IV-2619)
'defi'	atom 'defi' (IV-2528)
'desc'	atom 'desc' (IV-2528)
'dflt'	atom 'dflt' (IV-2529)
DialogPtr	"Graphics Types" (IV-2624)
DialogRef	"System and Toolbox Types" (IV-2638)
DigitizerInfo	struct DigitizerInfo (IV-2234)

'dimm'	atom 'dimm' (IV-2529)
'dinf'	atom 'dinf' (IV-2530)
DlgHookProcPtr	callback DlgHookProc (III-2079)
DlgHookUPP	"Universal Procedure Pointers" (IV-2641)
DlgHookYDProcPtr	callback DlgHookYDProc (III-2080)
DlgHookYDUPP	"Universal Procedure Pointers" (IV-2641)
'dmax'	atom 'dmax' (IV-2531)
'dmed'	atom 'dmed' (IV-2531)
DoMCActionProcPtr	callback DoMCActionProc (III-2082)
DoMCActionUPP	"Universal Procedure Pointers" (IV-2641)
DragConstraint	"Graphics Types" (IV-2624)
DragGrayRgnUPP	"Universal Procedure Pointers" (IV-2641)
'dref'	atom 'dref' (IV-2532)
'drep'	atom 'drep' (IV-2532)
EditListType	struct EditListType (IV-2241)
'edts'	atom 'edts' (IV-2533)
EffectsFrameParams	struct EffectsFrameParams (IV-2242)
EffectsFrameParamsPtr	"Effects Types" (IV-2622)
EffectSource	struct EffectSource (IV-2243)
EffectSourcePtr	"Effects Types" (IV-2622)
'elst'	atom 'elst' (IV-2533)
'end '	atom 'end ' (IV-2534)
'enda'	atom 'enda' (IV-2535)
EnteringNodeProcPtr	"Virtual Reality Types" (IV-2647)
EnteringNodeUPP	"Virtual Reality Types" (IV-2647)
EnumListRecord	struct EnumListRecord (IV-2244)
EnumRangeRecord	struct EnumRangeRecord (IV-2244)
EnumValuePair	struct EnumValuePair (IV-2245)
EQSpectrumBandsRecord	struct EQSpectrumBandsRecord (IV-2245)
EQSpectrumBandsRecordPtr	"Sound Types" (IV-2633)
EventKind	"System and Toolbox Types" (IV-2638)
EventMask	"System and Toolbox Types" (IV-2638)
EventModifiers	"System and Toolbox Types" (IV-2638)
EventRecord	struct EventRecord (IV-2246)
'expo'	atom 'expo' (IV-2535)
'ext '	atom 'ext ' (IV-2536)
ExtComponentResource	struct ExtComponentResource (IV-2249)
ExtComponentResourceHandle	"Component Types" (IV-2621)
ExtComponentResourcePtr	"Component Types" (IV-2621)
extended80	"Numeric Types" (IV-2631)
extended96	"Numeric Types" (IV-2631)
ExtendedScheduledSoundHeader	struct ExtendedScheduledSoundHeader (IV-2251)
ExtendedScheduledSoundHeaderPtr	"Sound Types" (IV-2633)
ExtendedSoundComponentData	struct ExtendedSoundComponentData (IV-2252)
ExtendedSoundComponentDataPtr	"Sound Types" (IV-2633)
ExtendedSoundParamBlock	struct ExtendedSoundParamBlock (IV-2254)
ExtendedSoundParamBlockPtr	"Sound Types" (IV-2633)
ExtSoundHeader	struct ExtSoundHeader (IV-2255)
ExtSoundHeaderPtr	"Sound Types" (IV-2633)

FieldInfoImageDescriptionExtension	struct FieldInfoImageDescriptionExtension (IV-2258)
FieldInfoImageDescriptionExtension2	struct FieldInfoImageDescriptionExtension2 (IV-2258)
FileFilterProcPtr	callback FileFilterProc (III-2082)
FileFilterUPP	"Universal Procedure Pointers" (IV-2641)
FileFilterYDProcPtr	callback FileFilterYDProc (III-2083)
FileFilterYDUPP	"Universal Procedure Pointers" (IV-2641)
FilePlayCompletionProcPtr	callback FilePlayCompletionProc (III-2084)
FilePlayCompletionUPP	"Universal Procedure Pointers" (IV-2641)
Fixed	"Numeric Types" (IV-2631)
FixedPoint	struct FixedPoint (IV-2258)
FixedPtr	"Numeric Types" (IV-2631)
FixedRangeRecord	struct FixedRangeRecord (IV-2259)
FixedRect	struct FixedRect (IV-2260)
'flap'	atom 'flap' (IV-2537)
FlashDescription	struct FlashDescription (IV-2260)
FlashDescriptionHandle	"Media Handling Types" (IV-2626)
FlashDescriptionPtr	"Media Handling Types" (IV-2626)
Float32	"Numeric Types" (IV-2631)
Float64	"Numeric Types" (IV-2631)
Float80	"Numeric Types" (IV-2631)
Float96	"Numeric Types" (IV-2631)
FloatPoint	"Virtual Reality Types" (IV-2647)
FourCharCode	"System and Toolbox Types" (IV-2638)
Fract	"Numeric Types" (IV-2631)
FractPtr	"Numeric Types" (IV-2631)
'free'	atom 'free' (IV-2538)
'frma'	atom 'frma' (IV-2538)
FSSpec	struct FSSpec (IV-2262)
FSSpecArrayPtr	"File System Types" (IV-2624)
FSSpecHandle	"File System Types" (IV-2624)
FSSpecPtr	"File System Types" (IV-2624)
'ftyp'	atom 'ftyp' (IV-2539)
GCInstrumentData	struct GCInstrumentData (IV-2263)
GCInstrumentDataHandle	"Music Types" (IV-2630)
GCInstrumentDataPtr	"Music Types" (IV-2630)
GCPart	struct GCPart (IV-2263)
GDevice	struct GDevice (IV-2264)
GDHandle	"Graphics Types" (IV-2624)
GDPtr	"Graphics Types" (IV-2624)
GenericKnobDescription	struct GenericKnobDescription (IV-2267)
GenericKnobDescriptionList	struct GenericKnobDescriptionList (IV-2268)
GenericKnobDescriptionListHandle	"Music Types" (IV-2630)
GenericKnobDescriptionListPtr	"Music Types" (IV-2630)
GetMissingComponentResourceProcPtr	callback GetMissingComponentResourceProc (III-2084)
GetMissingComponentResourceUPP	"Universal Procedure Pointers" (IV-2641)
GetMovieCompleteParams	struct GetMovieCompleteParams (IV-2268)

GetMovieProcPtr	callback GetMovieProc (III-2085)
GetMovieUPP	“Universal Procedure Pointers” (IV-2641)
'gmhd'	atom 'gmhd' (IV-2539)
'gmin'	atom 'gmin' (IV-2540)
GMinstrumentInfo	struct GMinstrumentInfo (IV-2272)
GMinstrumentInfoHandle	“Music Types” (IV-2630)
GMinstrumentInfoPtr	“Music Types” (IV-2630)
GradientColorPtr	“Effects Types” (IV-2622)
GradientColorRecord	struct GradientColorRecord (IV-2273)
GradientType	“Effects Types” (IV-2622)
GrafPort	struct GrafPort (IV-2273)
GrafPtr	“Graphics Types” (IV-2624)
GrafVars	“Graphics Types” (IV-2624)
GraphicImageMovieImportComponent	“Component Types” (IV-2621)
GraphicsExportComponent	“Component Types” (IV-2621)
GraphicsImportComponent	“Component Types” (IV-2621)
GWorldFlags	“Graphics Types” (IV-2624)
GWorldPtr	“Graphics Types” (IV-2624)
gxPath	struct gxPath (IV-2276)
gxPaths	struct gxPaths (IV-2277)
gxPoint	struct gxPoint (IV-2277)
Handle	“Memory Types” (IV-2627)
HandlerError	“Data Handling Types” (IV-2622)
'hdlr'	atom 'hdlr' (IV-2540)
HighHookUPP	“Universal Procedure Pointers” (IV-2641)
'hinf'	atom 'hinf' (IV-2541)
'hint'	atom 'hint' (IV-2542)
'hlit'	atom 'hlit' (IV-2543)
'hnti'	atom 'hnti' (IV-2544)
'hots'	atom 'hots' (IV-2544)
'hsin'	atom 'hsin' (IV-2545)
'hspa'	atom 'hspa' (IV-2545)
'htxt'	atom 'htxt' (IV-2546)
ICMAAlignmentProcPtr	callback ICMAAlignmentProc (III-2086)
ICMAAlignmentProcRecord	struct ICMAAlignmentProcRecord (IV-2278)
ICMAAlignmentProcRecordPtr	“Codec Types” (IV-2619)
ICMAAlignmentUPP	“Universal Procedure Pointers” (IV-2641)
ICMCompletionProcPtr	callback ICMCompletionProc (III-2087)
ICMCompletionProcRecord	struct ICMCompletionProcRecord (IV-2279)
ICMCompletionProcRecordPtr	“Codec Types” (IV-2619)
ICMCompletionUPP	“Universal Procedure Pointers” (IV-2641)
ICMConvertDataFormatProcPtr	callback ICMConvertDataFormatProc (III-2088)
ICMConvertDataFormatUPP	“Universal Procedure Pointers” (IV-2641)
ICMCursorNotify	“Codec Types” (IV-2619)
ICMCursorShieldedProcPtr	callback ICMCursorShieldedProc (III-2089)
ICMCursorShieldedUPP	“Universal Procedure Pointers” (IV-2641)
ICMDataProcPtr	callback ICMDataProc (III-2090)
ICMDataProcRecord	struct ICMDataProcRecord (IV-2279)
ICMDataProcRecordPtr	“Codec Types” (IV-2619)

ICMDataUPP	"Universal Procedure Pointers" (IV-2641)
ICMFlushProcPtr	callback ICMFlushProc (III-2091)
ICMFlushProcRecord	struct ICMFlushProcRecord (IV-2280)
ICMFlushProcRecordPtr	"Codec Types" (IV-2619)
ICMFlushUPP	"Universal Procedure Pointers" (IV-2641)
ICMFrameTimeInfo	struct ICMFrameTimeInfo (IV-2281)
ICMFrameTimeInfoPtr	"Codec Types" (IV-2619)
ICMFrameTimePtr	"Codec Types" (IV-2619)
ICMFrameTimeRecord	struct ICMFrameTimeRecord (IV-2281)
ICMMemoryDisposedProcPtr	callback ICMMemoryDisposedProc (III-2092)
ICMMemoryDisposedUPP	"Universal Procedure Pointers" (IV-2641)
ICMPixelFormatInfo	struct ICMPixelFormatInfo (IV-2282)
ICMPixelFormatInfoPtr	"Codec Types" (IV-2619)
ICMProgressProcPtr	callback ICMProgressProc (III-2093)
ICMProgressProcRecord	struct ICMProgressProcRecord (IV-2284)
ICMProgressProcRecordPtr	"Codec Types" (IV-2619)
ICMProgressUPP	"Universal Procedure Pointers" (IV-2641)
'idat'	atom 'idat' (IV-2546)
'idsc'	atom 'idsc' (IV-2547)
'iicc'	atom 'iicc' (IV-2547)
'imag'	atom 'imag' (IV-2547)
ImageCodecMPDrawBandProcPtr	callback ImageCodecMPDrawBandProc (III-2095)
ImageCodecMPDrawBandUPP	"Universal Procedure Pointers" (IV-2641)
ImageCodecTimeTriggerProcPtr	callback ImageCodecTimeTriggerProc (III-2096)
ImageCodecTimeTriggerUPP	"Universal Procedure Pointers" (IV-2641)
ImageDescription	struct ImageDescription (IV-2285)
ImageDescriptionHandle	"Codec Types" (IV-2619)
ImageDescriptionPtr	"Codec Types" (IV-2619)
ImageFieldSequence	"Codec Types" (IV-2619)
ImageRangeRecord	struct ImageRangeRecord (IV-2288)
ImageSequence	"Codec Types" (IV-2619)
ImageSequenceDataSource	"Codec Types" (IV-2619)
ImageSubCodecDecompressCapabilities	
	struct ImageSubCodecDecompressCapabilities (IV-2288)
ImageSubCodecDecompressRecord	
	struct ImageSubCodecDecompressRecord (IV-2289)
ImageTranscoderComponent	"Component Types" (IV-2621)
ImageTranscodeSequence	"Codec Types" (IV-2619)
ImagingCompleteProcPtr	"Virtual Reality Types" (IV-2647)
ImagingCompleteUPP	"Virtual Reality Types" (IV-2647)
'imap'	atom 'imap' (IV-2548)
'imct'	atom 'imct' (IV-2549)
'imda'	atom 'imda' (IV-2549)
'imgp'	atom 'imgp' (IV-2550)
'imgr'	atom 'imgr' (IV-2550)
'impn'	atom 'impn' (IV-2551)
'imre'	atom 'imre' (IV-2552)
'imrg'	atom 'imrg' (IV-2552)
'imrt'	atom 'imrt' (IV-2553)

' in'	atom ' in' (IV-2554)
InstCompInfo	struct InstCompInfo (IV-2291)
InstCompInfoHandle	"Music Types" (IV-2630)
InstCompInfoPtr	"Music Types" (IV-2630)
InstKnobList	struct InstKnobList (IV-2292)
InstKnobRec	struct InstKnobRec (IV-2293)
InstLibDescRec	struct InstLibDescRec (IV-2293)
InstrumentAboutInfo	struct InstrumentAboutInfo (IV-2293)
InstrumentAboutInfoHandle	"Music Types" (IV-2630)
InstrumentAboutInfoPtr	"Music Types" (IV-2630)
InstrumentInfoList	struct InstrumentInfoList (IV-2294)
InstrumentInfoListHandle	"Music Types" (IV-2630)
InstrumentInfoListPtr	"Music Types" (IV-2630)
InstrumentInfoRecord	struct InstrumentInfoRecord (IV-2295)
InstSampleDescRec	struct InstSampleDescRec (IV-2296)
ISAType	"System and Toolbox Types" (IV-2638)
ITab	struct ITab (IV-2297)
ITabHandle	"Graphics Types" (IV-2624)
ITabPtr	"Graphics Types" (IV-2624)
ItemCount	"System and Toolbox Types" (IV-2638)
KaraokeAtom	struct KaraokeAtom (IV-2298)
KaraokeRec	struct KaraokeRec (IV-2298)
'kmat'	atom 'kmat' (IV-2554)
KnobDescription	struct KnobDescription (IV-2299)
LangCode	"Text Types" (IV-2640)
LeavingNodeProcPtr	"Virtual Reality Types" (IV-2647)
LeavingNodeUPP	"Virtual Reality Types" (IV-2647)
LeftOverBlock	struct LeftOverBlock (IV-2301)
LeftOverBlockPtr	"Sound Types" (IV-2633)
LevelMeterInfo	struct LevelMeterInfo (IV-2301)
LevelMeterInfoPtr	"Sound Types" (IV-2633)
LHHandle	"Text Types" (IV-2640)
'link'	atom 'link' (IV-2556)
'load'	atom 'load' (IV-2556)
LogicalAddress	"Memory Types" (IV-2627)
LongRangeRecord	struct LongRangeRecord (IV-2302)
'LOOP'	atom 'LOOP' (IV-2557)
MacPolygon	struct MacPolygon (IV-2303)
MacRegion	struct MacRegion (IV-2303)
MatrixRecord	struct MatrixRecord (IV-2304)
MatrixRecordPtr	"Effects Types" (IV-2622)
'matt'	atom 'matt' (IV-2557)
'maxr'	atom 'maxr' (IV-2558)
mcAction	"Movie Controller Types" (IV-2628)
MCActionFilterProcPtr	callback MCActionFilterProc (III-2096)
MCActionFilterUPP	"Universal Procedure Pointers" (IV-2641)
MCActionFilterWithRefConProcPtr	callback MCActionFilterWithRefConProc (III-2097)
MCActionFilterWithRefConUPP	"Universal Procedure Pointers" (IV-2641)

mcFlags	"Movie Controller Types" (IV-2628)
MCInterfaceElement	"Movie Controller Types" (IV-2628)
'mdat'	atom 'mdat' (IV-2558)
'mdhd'	atom 'mdhd' (IV-2559)
'mdia'	atom 'mdia' (IV-2560)
Media	"Movie Types" (IV-2628)
MediaEQSpectrumBandsRecord	struct MediaEQSpectrumBandsRecord (IV-2304)
MediaEQSpectrumBandsRecordPtr	"Media Handling Types" (IV-2626)
MediaHandler	"Component Types" (IV-2621)
MediaHandlerComponent	"Component Types" (IV-2621)
mediaHandlerFlagsEnum	"Media Handling Types" (IV-2626)
MediaHeader	struct MediaHeader (IV-2305)
MediaPacketizerInfo	struct MediaPacketizerInfo (IV-2306)
MediaPacketizerInfoHandle	"Streaming Types" (IV-2636)
MediaPacketizerInfoPtr	"Streaming Types" (IV-2636)
MediaPacketizerRequirements	struct MediaPacketizerRequirements (IV-2308)
MediaPacketizerRequirementsPtr	"Streaming Types" (IV-2636)
MediaRecord	struct MediaRecord (IV-2310)
MenuHandle	"System and Toolbox Types" (IV-2638)
MenuInfo	struct MenuInfo (IV-2310)
MenuPtr	"System and Toolbox Types" (IV-2638)
'mime'	atom 'mime' (IV-2561)
'minf'(base)	atom 'minf'(base) (IV-2561)
'minf'(generic)	atom 'minf'(generic) (IV-2562)
'minf'(sound)	atom 'minf'(sound) (IV-2564)
'minf'(video)	atom 'minf'(video) (IV-2565)
ModalFilterProcPtr	callback ModalFilterProc (III-2098)
ModalFilterUPP	"Universal Procedure Pointers" (IV-2641)
ModalFilterYDProcPtr	callback ModalFilterYDProc (III-2099)
ModalFilterYDUPP	"Universal Procedure Pointers" (IV-2641)
ModifierTrackGraphicsModeRecord	struct ModifierTrackGraphicsModeRecord (IV-2311)
ModRef	struct ModRef (IV-2312)
'moov'	atom 'moov' (IV-2566)
MotionJPEGApp1Marker	struct MotionJPEGApp1Marker (IV-2312)
MouseOverHotSpotProcPtr	"Virtual Reality Types" (IV-2647)
MouseOverHotSpotUPP	"Virtual Reality Types" (IV-2647)
Movie	"Movie Types" (IV-2628)
MovieController	"Movie Controller Types" (IV-2628)
MovieControllerPtr	"Movie Controller Types" (IV-2628)
MovieDrawingCompleteProcPtr	callback MovieDrawingCompleteProc (III-2100)
MovieDrawingCompleteUPP	"Universal Procedure Pointers" (IV-2641)
MovieEditState	"Movie Types" (IV-2628)
MovieEditStateRecord	struct MovieEditStateRecord (IV-2313)
MovieExecuteWiredActionsProcPtr	callback MovieExecuteWiredActionsProc (III-2101)
MovieExecuteWiredActionsUPP	"Universal Procedure Pointers" (IV-2641)
MovieExportComponent	"Component Types" (IV-2621)
MovieExportGetDataParams	struct MovieExportGetDataParams (IV-2314)

MovieExportGetDataProcPtr	callback MovieExportGetDataProc (III-2101)
MovieExportGetDataUPP	"Universal Procedure Pointers" (IV-2641)
MovieExportGetPropertyProcPtr	callback MovieExportGetPropertyProc (III-2102)
MovieExportGetPropertyUPP	"Universal Procedure Pointers" (IV-2641)
movieFlattenFlagsEnum	"Movie Types" (IV-2628)
MovieHeader	struct MovieHeader (IV-2316)
MovieImportComponent	"Component Types" (IV-2621)
MovieMediaTimeRecord	struct MovieMediaTimeRecord (IV-2318)
MoviePrePrerollCompleteProcPtr	callback MoviePrePrerollCompleteProc (III-2104)
MoviePrePrerollCompleteUPP	"Universal Procedure Pointers" (IV-2641)
MoviePreviewCallOutProcPtr	callback MoviePreviewCallOutProc (III-2105)
MoviePreviewCallOutUPP	"Universal Procedure Pointers" (IV-2641)
MovieProgressProcPtr	callback MovieProgressProc (III-2105)
MovieProgressUPP	"Universal Procedure Pointers" (IV-2641)
MoviePtr	"Movie Types" (IV-2628)
MovieRecord	struct MovieRecord (IV-2318)
MovieRgnCoverProcPtr	callback MovieRgnCoverProc (III-2108)
MovieRgnCoverUPP	"Universal Procedure Pointers" (IV-2641)
MoviesErrorProcPtr	callback MoviesErrorProc (III-2109)
MoviesErrorUPP	"Universal Procedure Pointers" (IV-2641)
MoviesUserData	struct MoviesUserData (IV-2319)
MusicComponent	"Music Types" (IV-2630)
MusicController	"Music Types" (IV-2630)
MusicDescription	struct MusicDescription (IV-2319)
MusicDescriptionHandle	"Music Types" (IV-2630)
MusicDescriptionPtr	"Music Types" (IV-2630)
MusicMIDIPacket	struct MusicMIDIPacket (IV-2320)
MusicMIDIReadHookProcPtr	callback MusicMIDIReadHookProc (III-2109)
MusicMIDIReadHookUPP	"Universal Procedure Pointers" (IV-2641)
MusicMIDISendProcPtr	callback MusicMIDISendProc (III-2110)
MusicMIDISendUPP	"Universal Procedure Pointers" (IV-2641)
MusicOfflineDataProcPtr	callback MusicOfflineDataProc (III-2110)
MusicOfflineDataUPP	"Universal Procedure Pointers" (IV-2641)
MusicOpWord	"Music Types" (IV-2630)
MusicOpWordPtr	"Music Types" (IV-2630)
'mvhd'	atom 'mvhd' (IV-2567)
'name'(sprite)	atom 'name'(sprite) (IV-2568)
'name'(userdata)	atom 'name'(userdata) (IV-2569)
NCLCColorInfoImageDescriptionExtension	struct NCLCColorInfoImageDescriptionExtension (IV-2321)
'ndhd'	atom 'ndhd' (IV-2569)
nextTimeFlagsEnum	"Timing and Callback Types" (IV-2640)
'nloc'	atom 'nloc' (IV-2570)
nonGMInstrumentInfo	struct nonGMInstrumentInfo (IV-2321)
nonGMInstrumentInfoHandle	"Music Types" (IV-2630)
nonGMInstrumentInfoPtr	"Music Types" (IV-2630)
nonGMInstrumentInfoRecord	struct nonGMInstrumentInfoRecord (IV-2322)

NoteAllocator	"Music Types" (IV-2630)
NoteChannel	"Music Types" (IV-2630)
NoteRequest	struct NoteRequest (IV-2323)
NoteRequestInfo	struct NoteRequestInfo (IV-2323)
NullStHandle	"Text Types" (IV-2640)
'nump'	atom 'nump' (IV-2570)
NumVersion	struct NumVersion (IV-2324)
'obid'	atom 'obid' (IV-2571)
OfflineSampleType	struct OfflineSampleType (IV-2325)
OpenCPicParams	struct OpenCPicParams (IV-2326)
OptionBits	"System and Toolbox Types" (IV-2638)
OSErr	"System and Toolbox Types" (IV-2638)
OSStatus	"System and Toolbox Types" (IV-2638)
OSType	"System and Toolbox Types" (IV-2638)
OSTypePtr	"System and Toolbox Types" (IV-2638)
ParameterAlternateDataEntry	struct ParameterAlternateDataEntry (IV-2326)
ParameterAlternateDataType	struct ParameterAlternateDataType (IV-2327)
ParameterAtomTypeAndID	struct ParameterAtomTypeAndID (IV-2327)
ParameterDataBehavior	struct ParameterDataBehavior (IV-2328)
ParameterDataType	struct ParameterDataType (IV-2329)
ParameterDataUsage	struct ParameterDataUsage (IV-2330)
ParameterDependancyRecord	struct ParameterDependancyRecord (IV-2330)
Pattern	struct Pattern (IV-2330)
'payt'	atom 'payt' (IV-2571)
PBVersion	"System and Toolbox Types" (IV-2638)
'pdat'	atom 'pdat' (IV-2572)
PhysicalAddress	"Memory Types" (IV-2627)
PicHandle	"Graphics Types" (IV-2624)
PicPtr	"Graphics Types" (IV-2624)
Picture	struct Picture (IV-2331)
PixelAspectRatioImageDescriptionExtension	struct PixelAspectRatioImageDescriptionExtension (IV-2332)
PixMap	struct PixMap (IV-2332)
PixMapExtension	struct PixMapExtension (IV-2335)
PixMapExtHandle	"Graphics Types" (IV-2624)
PixMapExtPtr	"Graphics Types" (IV-2624)
PixMapHandle	"Graphics Types" (IV-2624)
PixMapPtr	"Graphics Types" (IV-2624)
PixPat	struct PixPat (IV-2336)
PixPatHandle	"Graphics Types" (IV-2624)
PixPatPtr	"Graphics Types" (IV-2624)
PlanarComponentInfo	struct PlanarComponentInfo (IV-2337)
PlanarPixMapInfo	struct PlanarPixMapInfo (IV-2337)
PlanarPixmapInfoSorensonYUV9	struct PlanarPixmapInfoSorensonYUV9 (IV-2338)
PlanarPixmapInfoYUV420	struct PlanarPixmapInfoYUV420 (IV-2338)
playHintsEnum	"Movie Types" (IV-2628)
'pmax'	atom 'pmax' (IV-2572)
'pnot'	atom 'pnot' (IV-2573)
pnotComponent	"Component Types" (IV-2621)

Point	struct Point (IV-2339)
PointPtr	"Memory Types" (IV-2627)
PolyHandle	"Graphics Types" (IV-2624)
PolyPtr	"Graphics Types" (IV-2624)
PrePrerollCompleteProcPtr	callback PrePrerollCompleteProc (III-2111)
PrePrerollCompleteUPP	"Universal Procedure Pointers" (IV-2641)
PreviewResource	"Codec Types" (IV-2619)
PreviewResourcePtr	"Codec Types" (IV-2619)
PreviewResourceRecord	struct PreviewResourceRecord (IV-2340)
ProcHandle	"Universal Procedure Pointers" (IV-2641)
ProcInfoType	"System and Toolbox Types" (IV-2638)
ProcPtr	callback Proc (III-2112)
Ptr	"Memory Types" (IV-2627)
PublicHandlerInfo	struct PublicHandlerInfo (IV-2341)
QDArcProcPtr	callback QDArcProc (III-2112)
QDArcUPP	"Universal Procedure Pointers" (IV-2641)
QDBitsProcPtr	callback QDBitsProc (III-2113)
QDBitsUPP	"Universal Procedure Pointers" (IV-2641)
QDCommentProcPtr	callback QDCommentProc (III-2114)
QDCommentUPP	"Universal Procedure Pointers" (IV-2641)
QDGetPicProcPtr	callback QDGetPicProc (III-2114)
QDGetPicUPP	"Universal Procedure Pointers" (IV-2641)
QDJShieldCursorProcPtr	callback QDJShieldCursorProc (III-2115)
QDJShieldCursorUPP	"Universal Procedure Pointers" (IV-2641)
QDLineProcPtr	callback QDLineProc (III-2115)
QDLineUPP	"Universal Procedure Pointers" (IV-2641)
QDOpcodeProcPtr	callback QDOpcodeProc (III-2116)
QDOpcodeUPP	"Universal Procedure Pointers" (IV-2641)
QDOvalProcPtr	callback QDOvalProc (III-2116)
QDOvalUPP	"Universal Procedure Pointers" (IV-2641)
QDPixProcPtr	callback QDPixProc (III-2117)
QDPixUPP	"Universal Procedure Pointers" (IV-2641)
QDPolyProcPtr	callback QDPolyProc (III-2118)
QDPolyUPP	"Universal Procedure Pointers" (IV-2641)
QDPrinterStatusUPP	"Universal Procedure Pointers" (IV-2641)
QDProcs	struct QDProcs (IV-2342)
QDProcsPtr	"Graphics Types" (IV-2624)
QDPutPicProcPtr	callback QDPutPicProc (III-2119)
QDPutPicUPP	"Universal Procedure Pointers" (IV-2641)
QDRectProcPtr	callback QDRectProc (III-2119)
QDRectUPP	"Universal Procedure Pointers" (IV-2641)
'qdrg'	atom 'qdrg' (IV-2574)
QDRgnProcPtr	callback QDRgnProc (III-2120)
QDRgnUPP	"Universal Procedure Pointers" (IV-2641)
QDRRectProcPtr	callback QDRRectProc (III-2121)
QDRRectUPP	"Universal Procedure Pointers" (IV-2641)
QDStdGlyphsProcPtr	callback QDStdGlyphsProc (III-2122)
QDStdGlyphsUPP	"Universal Procedure Pointers" (IV-2641)
QDTextProcPtr	callback QDTextProc (III-2122)

QDTextUPP	“Universal Procedure Pointers” (IV-2641)
QDTxMeasProcPtr	callback QDTxMeasProc (III-2123)
QDTxMeasUPP	“Universal Procedure Pointers” (IV-2641)
QElem	struct QElem (IV-2344)
QElemPtr	“Memory Types” (IV-2627)
QHdr	struct QHdr (IV-2345)
QHdrPtr	“Memory Types” (IV-2627)
QTAltComponentCheckRecord	struct QTAltComponentCheckRecord (IV-2345)
QTAltCPURatingRecord	struct QTAltCPURatingRecord (IV-2346)
QTAltDataRateRecord	struct QTAltDataRateRecord (IV-2347)
QTAltLanguageRecord	struct QTAltLanguageRecord (IV-2348)
QTAltVersionCheckRecord	struct QTAltVersionCheckRecord (IV-2348)
QAtom	“Atom Types” (IV-2619)
QAtomContainer	“Atom Types” (IV-2619)
QAtomID	“Atom Types” (IV-2619)
QAtomSpec	struct QAtomSpec (IV-2349)
QAtomSpecPtr	“Media Handling Types” (IV-2626)
QAtomType	“Atom Types” (IV-2619)
QTBandwidthNotificationProcPtr	callback QTBandwidthNotificationProc (III-2124)
QTBandwidthNotificationUPP	“Universal Procedure Pointers” (IV-2641)
QTBandwidthReference	“Movie Types” (IV-2628)
QTCallBack	“Timing and Callback Types” (IV-2640)
QTCallBackFlags	“Timing and Callback Types” (IV-2640)
QTCallBackHeader	struct QTCallBackHeader (IV-2349)
QTCallBackProcPtr	callback QTCallBackProc (III-2124)
QTCallBackType	“Timing and Callback Types” (IV-2640)
QTCallBackUPP	“Universal Procedure Pointers” (IV-2641)
QTChapterInfoPtr	“Movie Controller Types” (IV-2628)
QTChapterInfoRecord	struct QTChapterInfoRecord (IV-2351)
QTCustomActionTargetPtr	“Media Handling Types” (IV-2626)
QTCustomActionTargetRecord	struct QTCustomActionTargetRecord (IV-2351)
QDoScriptPtr	“Movie Controller Types” (IV-2628)
QDoScriptRecord	struct QDoScriptRecord (IV-2352)
QTEffectListOptions	“Effects Types” (IV-2622)
QTEvaluateExpressionPtr	“Movie Controller Types” (IV-2628)
QTEvaluateExpressionRecord	struct QTEvaluateExpressionRecord (IV-2353)
QTEventRecord	struct QTEventRecord (IV-2353)
QTEventRecordPtr	“Media Handling Types” (IV-2626)
QTFetchParameterAsPtr	“Movie Controller Types” (IV-2628)
QTFetchParameterAsRecord	struct QTFetchParameterAsRecord (IV-2354)
QTFloatDouble	“Numeric Types” (IV-2631)
QTFloatSingle	“Numeric Types” (IV-2631)
QTGetChapterTimePtr	“Movie Controller Types” (IV-2628)
QTGetChapterTimeRecord	struct QTGetChapterTimeRecord (IV-2355)
QTGetCursorByIDPtr	“Movie Controller Types” (IV-2628)
QTGetCursorByIDRecord	struct QTGetCursorByIDRecord (IV-2356)
QTGetExternalMoviePtr	“Movie Controller Types” (IV-2628)
QTGetExternalMovieRecord	struct QTGetExternalMovieRecord (IV-2356)

QTMIDIComponent		“Music Types” (IV-2630)
QTMIDIPort	struct QTMIDIPort	(IV-2357)
QTMIDIPortList	struct QTMIDIPortList	(IV-2357)
QTMIDIPortListHandle		“Music Types” (IV-2630)
QTMIDIPortListPtr		“Music Types” (IV-2630)
QTMLMutex	“Windows Programming Types”	(IV-2648)
QTMLSyncVar	“Windows Programming Types”	(IV-2648)
QTMLSyncVarPtr	“Windows Programming Types”	(IV-2648)
QTMovieExportSourceInfo	struct QTMovieExportSourceInfo	(IV-2358)
QTMovieExportSourceRecord	struct QTMovieExportSourceRecord	(IV-2358)
QTParamDialogEventPtr		“Effects Types” (IV-2622)
QTParamDialogEventRecord	struct QTParamDialogEventRecord	(IV-2359)
QTParameterDialog		“Effects Types” (IV-2622)
QTParameterDialogOptions		“Effects Types” (IV-2622)
QTParameterValidationOptions		“Effects Types” (IV-2622)
QTParamFetchPreviewPtr		“Effects Types” (IV-2622)
QTParamFetchPreviewRecord	struct QTParamFetchPreviewRecord	(IV-2360)
QTParamPreviewPtr		“Effects Types” (IV-2622)
QTParamPreviewRecord	struct QTParamPreviewRecord	(IV-2360)
QTPresetInfo	struct QTPresetInfo	(IV-2360)
QTPresetListRecord	struct QTPresetListRecord	(IV-2361)
QTResolutionSettings	struct QTResolutionSettings	(IV-2362)
QTRestartAtTimePtr		“Movie Controller Types” (IV-2628)
QTRestartAtTimeRecord	struct QTRestartAtTimeRecord	(IV-2362)
QTRuntimeSpriteDescPtr		“Sprite Management Types” (IV-2636)
QTRuntimeSpriteDescStruct	struct QTRuntimeSpriteDescStruct	(IV-2363)
QTSAtomContainerDataStruct	struct QTSAtomContainerDataStruct	(IV-2364)
QTSAudioParams	struct QTSAudioParams	(IV-2364)
QTSBandwidthAlertParams	struct QTSBandwidthAlertParams	(IV-2365)
QTSBufferTimeAtom	struct QTSBufferTimeAtom	(IV-2366)
QTSCanHandleSendDataParams		
	struct QTSCanHandleSendDataTypeParams	(IV-2366)
QTScheduledBandwidthHandle		“Movie Types” (IV-2628)
QTScheduledBandwidthPtr		“Movie Types” (IV-2628)
QTScheduledBandwidthRecord	struct QTScheduledBandwidthRecord	(IV-2366)
QTScheduledBandwidthReference		“Movie Types” (IV-2628)
QTSClipRectAtom	struct QTSClipRectAtom	(IV-2367)
QTSDataRateHistoryParams	struct QTSDataRateHistoryParams	(IV-2368)
QTSDimensionParams	struct QTSDimensionParams	(IV-2368)
QTSDirectConnectPrefsRecord	struct QTSDirectConnectPrefsRecord	(IV-2369)
QTSDurationAtom	struct QTSDurationAtom	(IV-2369)
QTSEditEntry	struct QTSEditEntry	(IV-2370)
QTSEditList	struct QTSEditList	(IV-2370)
QTSEditListHandle		“Streaming Types” (IV-2636)
QTSEditListPtr		“Streaming Types” (IV-2636)
QTSErrorParams	struct QTSErrorParams	(IV-2371)
QTSettingsVersionAtomRecord	struct QTSettingsVersionAtomRecord	(IV-2371)
QTSGetURLLinkRecord	struct QTSGetURLLinkRecord	(IV-2371)
QTSGraphicsModeParams	struct QTSGraphicsModeParams	(IV-2372)

QTInfoParams	struct QTInfoParams	(IV-2372)
QTSLostPercentParams	struct QTSLostPercentParams	(IV-2373)
QTSMediaIndSampleDescriptionParams	struct QTSMediaIndSampleDescriptionParams	(IV-2373)
QTSMediaNotificationParams	struct QTSMediaNotificationParams	(IV-2374)
QTSMediaParams	struct QTSMediaParams	(IV-2374)
QTSMediaPresentationParams	struct QTSMediaPresentationParams	(IV-2375)
QTSMediaStreamHeaderAtom	struct QTSMediaStreamHeaderAtom	(IV-2375)
QTSMemPtr	"Streaming Types"	(IV-2636)
QTSNameParams	struct QTSNameParams	(IV-2376)
QTSNewPresDetectedParams	struct QTSNewPresDetectedParams	(IV-2376)
QTSNewPresentationParams	struct QTSNewPresentationParams	(IV-2376)
QTSNewStreamParams	struct QTSNewStreamParams	(IV-2377)
QTSNoProxyPref	struct QTSNoProxyPref	(IV-2378)
QTSNotificationProcPtr	callback QTSNotificationProc	(III-2126)
QTSNotificationUPP	"Universal Procedure Pointers"	(IV-2641)
QTSPresentation	"Streaming Types"	(IV-2636)
QTSPresentationHeaderAtom	struct QTSPresentationHeaderAtom	(IV-2378)
QTSPresentationRecord	struct QTSPresentationRecord	(IV-2379)
QTSPresIdleParams	struct QTSPresIdleParams	(IV-2379)
QTSpriteButtonBehaviorPtr	"Sprite Management Types"	(IV-2636)
QTSpriteButtonBehaviorStruct	struct QTSpriteButtonBehaviorStruct	(IV-2380)
QTSProxyPref	struct QTSProxyPref	(IV-2380)
QTSProxyPrefsRecord	struct QTSProxyPrefsRecord	(IV-2381)
QTSSampleDescription	struct QTSSampleDescription	(IV-2381)
QTSSampleDescriptionHandle	"Streaming Types"	(IV-2636)
QTSSampleDescriptionPtr	"Streaming Types"	(IV-2636)
QTSStatHelper	"Streaming Types"	(IV-2636)
QTSStatHelperNextParams	struct QTSStatHelperNextParams	(IV-2382)
QTSStatHelperRecord	struct QTSStatHelperRecord	(IV-2384)
QTSStatisticsParams	struct QTSStatisticsParams	(IV-2384)
QTSStatus	"Streaming Types"	(IV-2636)
QTSStatusParams	struct QTSStatusParams	(IV-2385)
QTSStream	"Streaming Types"	(IV-2636)
QTSStreamBuffer	struct QTSStreamBuffer	(IV-2385)
QTSStreamChangedParams	struct QTSStreamChangedParams	(IV-2386)
QTSStreamGoneParams	struct QTSStreamGoneParams	(IV-2387)
QTSStreamRecord	struct QTSStreamRecord	(IV-2387)
QTStatusStringPtr	"Movie Controller Types"	(IV-2628)
QTStatusStringRecord	struct QTStatusStringRecord	(IV-2387)
QTSTransportPref	struct QTSTransportPref	(IV-2388)
QTSURLParams	struct QTSURLParams	(IV-2389)
QTSVideoParams	struct QTSVideoParams	(IV-2389)
QTSVolumesParams	struct QTSVolumesParams	(IV-2390)
QTSyncTaskProcPtr	callback QTSyncTaskProc	(III-2127)
QTSyncTaskPtr	"Timing and Callback Types"	(IV-2640)
QTSyncTaskRecord	struct QTSyncTaskRecord	(IV-2391)
QTSyncTaskUPP	"Universal Procedure Pointers"	(IV-2641)
QTTargetDataSize	struct QTTargetDataSize	(IV-2391)

QTTweener	"Component Types" (IV-2621)
QTTweenerRecord	struct QTTweenerRecord (IV-2391)
QTVideoOutputComponent	"Video Types" (IV-2646)
QTVRAngularUnits	"Virtual Reality Types" (IV-2647)
QTVRAreaOfInterest	struct QTVRAreaOfInterest (IV-2392)
QTVRBackBufferImagingProcPtr	callback QTVRBackBufferImagingProc (III-2127)
QTVRBackBufferImagingUPP	"Universal Procedure Pointers" (IV-2641)
QTVRControlSetting	"Virtual Reality Types" (IV-2647)
QTVRCubicFaceData	struct QTVRCubicFaceData (IV-2393)
QTVRCubicViewAtom	struct QTVRCubicViewAtom (IV-2394)
QTVRCursorRecord	struct QTVRCursorRecord (IV-2395)
QTVREnteringNodeProcPtr	callback QTVREnteringNodeProc (III-2128)
QTVREnteringNodeUPP	"Universal Procedure Pointers" (IV-2641)
QTVRFloatPoint	struct QTVRFloatPoint (IV-2396)
QTVRImagingCompleteProcPtr	callback QTVRImagingCompleteProc (III-2129)
QTVRImagingCompleteUPP	"Universal Procedure Pointers" (IV-2641)
QTVRImagingMode	"Virtual Reality Types" (IV-2647)
QTVRInstance	"Virtual Reality Types" (IV-2647)
QTVRInterceptProcPtr	callback QTVRInterceptProc (III-2130)
QTVRInterceptPtr	"Virtual Reality Types" (IV-2647)
QTVRInterceptRecord	struct QTVRInterceptRecord (IV-2396)
QTVRInterceptUPP	"Universal Procedure Pointers" (IV-2641)
QTVRLeavingNodeProcPtr	callback QTVRLeavingNodeProc (III-2130)
QTVRLeavingNodeUPP	"Universal Procedure Pointers" (IV-2641)
QTVRMouseOverHotSpotProcPtr	callback QTVRMouseOverHotSpotProc (III-2131)
QTVRMouseOverHotSpotUPP	"Universal Procedure Pointers" (IV-2641)
QTVRNudgeControl	"Virtual Reality Types" (IV-2647)
QTVRNudgeMode	"Virtual Reality Types" (IV-2647)
QTVRObjectAnimationSetting	"Virtual Reality Types" (IV-2647)
QTVRPanoImagingAtom	struct QTVRPanoImagingAtom (IV-2398)
QTVRProcSelector	"Virtual Reality Types" (IV-2647)
QTVRViewStateType	"Virtual Reality Types" (IV-2647)
RDFFlagsType	"System and Toolbox Types" (IV-2638)
'rdrf'	atom 'rdrf' (IV-2575)
Rect	struct Rect (IV-2399)
RectPtr	"Memory Types" (IV-2627)
ReferenceMovieDataRefRecord	struct ReferenceMovieDataRefRecord (IV-2400)
RegionCode	"Text Types" (IV-2640)
RegisteredComponentInstancePtr	"Component Types" (IV-2621)
RegisteredComponentInstanceRecord	struct RegisteredComponentInstanceRecord (IV-2401)
RegisteredComponentInstanceRecordPtr	"Component Types" (IV-2621)
RegisteredComponentPtr	"Component Types" (IV-2621)
RegisteredComponentRecord	struct RegisteredComponentRecord (IV-2401)
RegisteredComponentRecordPtr	"Component Types" (IV-2621)
'reso'	atom 'reso' (IV-2576)
ResolvedQTEventSpec	struct ResolvedQTEventSpec (IV-2401)
ResolvedQTEventSpecPtr	"Media Handling Types" (IV-2626)
ResourceSpec	struct ResourceSpec (IV-2402)

ResTypePtr	"System and Toolbox Types" (IV-2638)
RGBColor	struct RGBColor (IV-2403)
RGBColorHdl	"Graphics Types" (IV-2624)
RGBColorPtr	"Graphics Types" (IV-2624)
RGBRangeRecord	struct RGBRangeRecord (IV-2403)
RgnHandle	"Memory Types" (IV-2627)
RgnPtr	"Memory Types" (IV-2627)
'rmcd'	atom 'rmcd' (IV-2576)
'rmcs'	atom 'rmcs' (IV-2577)
'rmda'	atom 'rmda' (IV-2577)
'rmdr'	atom 'rmdr' (IV-2578)
'rmla'	atom 'rmla' (IV-2579)
'rmqu'	atom 'rmqu' (IV-2579)
'rmra'	atom 'rmra' (IV-2580)
'rmvc'	atom 'rmvc' (IV-2581)
RoutineDescriptor	struct RoutineDescriptor (IV-2404)
RoutineFlagsType	"System and Toolbox Types" (IV-2638)
RoutineRecord	struct RoutineRecord (IV-2405)
RTPMediaPacketizer	"Streaming Types" (IV-2636)
RTPMPDataReleaseProcPtr	callback RTPMPDataReleaseProc (III-2132)
RTPMPDataReleaseUPP	"Universal Procedure Pointers" (IV-2641)
RTPMPPayloadTypeParams	struct RTPMPPayloadTypeParams (IV-2407)
RTPMPSampleDataParams	struct RTPMPSampleDataParams (IV-2407)
RTPMPSampleRef	"Streaming Types" (IV-2636)
RTPPacketBuilder	"Streaming Types" (IV-2636)
RTPPacketGroupRef	"Streaming Types" (IV-2636)
RTPPacketRef	"Streaming Types" (IV-2636)
RTPPacketRepeatedDataRef	"Streaming Types" (IV-2636)
RTPPayloadCharacteristic	struct RTPPayloadCharacteristic (IV-2409)
RTPPayloadInfo	struct RTPPayloadInfo (IV-2409)
RTPPayloadInfoHandle	"Streaming Types" (IV-2636)
RTPPayloadInfoPtr	"Streaming Types" (IV-2636)
RTPPayloadSortRequest	struct RTPPayloadSortRequest (IV-2410)
RTPPayloadSortRequestPtr	"Streaming Types" (IV-2636)
RTPPBCallbackProcPtr	callback RTPPBCallbackProc (III-2133)
RTPPBCallbackUPP	"Universal Procedure Pointers" (IV-2641)
RTPReassembler	"Streaming Types" (IV-2636)
RTPReassemblerInfo	struct RTPReassemblerInfo (IV-2410)
RTPReassemblerInfoHandle	"Streaming Types" (IV-2636)
RTPReassemblerInfoPtr	"Streaming Types" (IV-2636)
RTPRssmInitParams	struct RTPRssmInitParams (IV-2411)
RTPRssmPacket	struct RTPRssmPacket (IV-2412)
RTPSendStreamBufferRangeParams	struct RTPSendStreamBufferRangeParams (IV-2413)
RTPSSRC	"Streaming Types" (IV-2636)
SampleDescription	struct SampleDescription (IV-2414)
SampleDescriptionHandle	"Media Handling Types" (IV-2626)
SampleDescriptionPtr	"Media Handling Types" (IV-2626)
SampleReference64Ptr	"Media Handling Types" (IV-2626)

SampleReference64Record	struct SampleReference64Record (IV-2415)
SampleReferencePtr	“Media Handling Types” (IV-2626)
SampleReferenceRecord	struct SampleReferenceRecord (IV-2416)
SampleToChunk	struct SampleToChunk (IV-2417)
SCDataRateSettings	struct SCDataRateSettings (IV-2417)
SCEExtendedProcs	struct SCEExtendedProcs (IV-2418)
ScheduledSoundHeader	struct ScheduledSoundHeader (IV-2419)
ScheduledSoundHeaderPtr	“Sound Types” (IV-2633)
SCModalFilterProcPtr	callback SCModalFilterProc (III-2133)
SCModalFilterUPP	“Universal Procedure Pointers” (IV-2641)
SCModalHookProcPtr	callback SCModalHookProc (III-2134)
SCModalHookUPP	“Universal Procedure Pointers” (IV-2641)
SCParams	struct SCParams (IV-2420)
'scpt'	atom 'scpt' (IV-2581)
ScriptCode	“Text Types” (IV-2640)
ScrpSTElement	struct ScrpSTElement (IV-2422)
SCSpatialSettings	struct SCSpatialSettings (IV-2423)
SCStatus	struct SCStatus (IV-2425)
SCStatusPtr	“Sound Types” (IV-2633)
SCTemporalSettings	struct SCTemporalSettings (IV-2427)
'sdp '	atom 'sdp ' (IV-2582)
'sean'	atom 'sean' (IV-2582)
'se10'	atom 'se10' (IV-2582)
SeqGrabChannelInfoEnum	“Sequence Grabbing Types” (IV-2632)
SeqGrabComponent	“Sequence Grabbing Types” (IV-2632)
SeqGrabDataOutputEnum	“Sequence Grabbing Types” (IV-2632)
SeqGrabExtendedFrameInfo	struct SeqGrabExtendedFrameInfo (IV-2429)
SeqGrabExtendedFrameInfoPtr	“Sequence Grabbing Types” (IV-2632)
SeqGrabFrameInfo	struct SeqGrabFrameInfo (IV-2430)
SeqGrabFrameInfoPtr	“Sequence Grabbing Types” (IV-2632)
SeqGrabUsageEnum	“Sequence Grabbing Types” (IV-2632)
SequenceFilterComponent	“Component Types” (IV-2621)
SequenceFilterDataProcPtr	callback SequenceFilterDataProc (III-2135)
SequenceFilterDataUPP	“Universal Procedure Pointers” (IV-2641)
SFModalFilterProcPtr	callback SFModalFilterProc (III-2136)
SFModalFilterUPP	“Universal Procedure Pointers” (IV-2641)
SFReply	struct SFReply (IV-2431)
SGAddFrameBottleProcPtr	callback SGAddFrameBottleProc (III-2136)
SGAddFrameBottleUPP	“Universal Procedure Pointers” (IV-2641)
SGChannel	“Sequence Grabbing Types” (IV-2632)
SGCompressBottleProcPtr	callback SGCompressBottleProc (III-2137)
SGCompressBottleUPP	“Universal Procedure Pointers” (IV-2641)
SGCompressCompleteBottleProcPtr	callback SGCompressCompleteBottleProc (III-2138)
SGCompressCompleteBottleUPP	“Universal Procedure Pointers” (IV-2641)
SGCompressInfo	struct SGCompressInfo (IV-2432)
SGDataProcPtr	callback SGDataProc (III-2139)
SGDataUPP	“Universal Procedure Pointers” (IV-2641)
SGDeviceList	“Sequence Grabbing Types” (IV-2632)

SGDeviceListPtr	"Sequence Grabbing Types" (IV-2632)
SGDeviceListRecord	struct SGDeviceListRecord (IV-2433)
SGDeviceName	struct SGDeviceName (IV-2434)
SGDisplayBottleProcPtr	callback SGDisplayBottleProc (III-2140)
SGDisplayBottleUPP	"Universal Procedure Pointers" (IV-2641)
SGDisplayCompressBottleProcPtr	callback SGDisplayCompressBottleProc (III-2141)
SGDisplayCompressBottleUPP	"Universal Procedure Pointers" (IV-2641)
SGGrabBottleProcPtr	callback SGGrabBottleProc (III-2142)
SGGrabBottleUPP	"Universal Procedure Pointers" (IV-2641)
SGGrabCompleteBottleProcPtr	callback SGGrabCompleteBottleProc (III-2143)
SGGrabCompleteBottleUPP	"Universal Procedure Pointers" (IV-2641)
SGGrabCompressCompleteBottleProcPtr	callback SGGrabCompressCompleteBottleProc (III-2143)
SGGrabCompressCompleteBottleUPP	"Universal Procedure Pointers" (IV-2641)
SGModalFilterProcPtr	callback SGModalFilterProc (III-2144)
SGModalFilterUPP	"Universal Procedure Pointers" (IV-2641)
SGOutput	"Sequence Grabbing Types" (IV-2632)
SGOutputRecord	struct SGOutputRecord (IV-2435)
SGTransferFrameBottleProcPtr	callback SGTransferFrameBottleProc (III-2145)
SGTransferFrameBottleUPP	"Universal Procedure Pointers" (IV-2641)
ShadowSync	struct ShadowSync (IV-2436)
SHChunkRecord	struct SHChunkRecord (IV-2436)
ShortFixed	"Numeric Types" (IV-2631)
SHServerEditParameters	struct SHServerEditParameters (IV-2437)
SICCompletionProcPtr	callback SICCompletionProc (III-2146)
SICCompletionProcPtr	callback SICCompletionProc (III-2146)
SICCompletionUPP	"Universal Procedure Pointers" (IV-2641)
SignedByte	"Numeric Types" (IV-2631)
SIInterruptProcPtr	callback SIInterruptProc (III-2147)
SIInterruptUPP	"Universal Procedure Pointers" (IV-2641)
SInt16	"Numeric Types" (IV-2631)
SInt32	"Numeric Types" (IV-2631)
SInt64	"Numeric Types" (IV-2631)
SInt8	"Numeric Types" (IV-2631)
Size	"Memory Types" (IV-2627)
'skip'	atom 'skip' (IV-2583)
'smhd'	atom 'smhd' (IV-2584)
SMPTEFlags	"Effects Types" (IV-2622)
SMPTEFrameReference	"Effects Types" (IV-2622)
SMPTEWipeType	"Effects Types" (IV-2622)
SMStatus	struct SMStatus (IV-2438)
SMStatusPtr	"Sound Types" (IV-2633)
Snd2ListHandle	"Sound Types" (IV-2633)
Snd2ListHndl	"Sound Types" (IV-2633)
Snd2ListPtr	"Sound Types" (IV-2633)
Snd2ListResource	struct Snd2ListResource (IV-2439)
SndCallBackProcPtr	callback SndCallBackProc (III-2147)
SndCallBackUPP	"Universal Procedure Pointers" (IV-2641)

SndChannel	struct SndChannel (IV-2440)
SndChannelPtr	"Sound Types" (IV-2633)
SndCommand	struct SndCommand (IV-2441)
SndDoubleBackProcPtr	callback SndDoubleBackProc (III-2148)
SndDoubleBackUPP	"Universal Procedure Pointers" (IV-2641)
SndDoubleBuffer	struct SndDoubleBuffer (IV-2442)
SndDoubleBufferHeader	struct SndDoubleBufferHeader (IV-2443)
SndDoubleBufferHeader2	struct SndDoubleBufferHeader2 (IV-2444)
SndDoubleBufferHeader2Ptr	"Sound Types" (IV-2633)
SndDoubleBufferHeaderPtr	"Sound Types" (IV-2633)
SndDoubleBufferPtr	"Sound Types" (IV-2633)
SndInputCmpParam	struct SndInputCmpParam (IV-2446)
SndInputCmpParamPtr	"Sound Types" (IV-2633)
SndListHandle	"Sound Types" (IV-2633)
SndListHndl	"Sound Types" (IV-2633)
SndListPtr	"Sound Types" (IV-2633)
SndListResource	struct SndListResource (IV-2447)
SoundComponentData	struct SoundComponentData (IV-2448)
SoundComponentDataPtr	"Sound Types" (IV-2633)
SoundComponentLink	struct SoundComponentLink (IV-2449)
SoundComponentLinkPtr	"Sound Types" (IV-2633)
SoundConverter	"Sound Types" (IV-2633)
SoundConverterFillBufferDataProcPtr	callback SoundConverterFillBufferDataProc (III-2148)
SoundConverterFillBufferDataUPP	"Universal Procedure Pointers" (IV-2641)
SoundDescription	struct SoundDescription (IV-2449)
SoundDescriptionHandle	"Sound Types" (IV-2633)
SoundDescriptionPtr	"Sound Types" (IV-2633)
SoundDescriptionV1	struct SoundDescriptionV1 (IV-2451)
SoundDescriptionV1Handle	"Sound Types" (IV-2633)
SoundDescriptionV1Ptr	"Sound Types" (IV-2633)
SoundHeader	struct SoundHeader (IV-2452)
SoundHeaderPtr	"Sound Types" (IV-2633)
SoundHeaderUnion	"Sound Types" (IV-2633)
SoundInfoList	struct SoundInfoList (IV-2453)
SoundInfoListPtr	"Sound Types" (IV-2633)
SoundMediaInfoHeader	struct SoundMediaInfoHeader (IV-2453)
SoundParamBlock	struct SoundParamBlock (IV-2454)
SoundParamBlockPtr	"Sound Types" (IV-2633)
SoundParamProcPtr	callback SoundParamProc (III-2149)
SoundParamUPP	"Universal Procedure Pointers" (IV-2641)
SoundSlopeAndInterceptPtr	"Sound Types" (IV-2633)
SoundSlopeAndInterceptRecord	struct SoundSlopeAndInterceptRecord (IV-2456)
SoundSource	"Sound Types" (IV-2633)
SoundSourcePtr	"Sound Types" (IV-2633)
SourceData	"Codec Types" (IV-2619)
SPB	struct SPB (IV-2456)
SPBPtr	"Sound Types" (IV-2633)
Sprite	"Sprite Management Types" (IV-2636)

SpriteDescription	struct SpriteDescription (IV-2458)
SpriteDescriptionHandle	“Sprite Management Types” (IV-2636)
SpriteDescriptionPtr	“Sprite Management Types” (IV-2636)
SpriteRecord	struct SpriteRecord (IV-2459)
SpriteWorld	“Sprite Management Types” (IV-2636)
SpriteWorldRecord	struct SpriteWorldRecord (IV-2459)
SProcHndl	“System and Toolbox Types” (IV-2638)
SProcPtr	“System and Toolbox Types” (IV-2638)
SProcRec	struct SProcRec (IV-2459)
'sprt'	atom 'sprt' (IV-2584)
'sptl'	atom 'sptl' (IV-2586)
SSpLocalizationData	struct SSpLocalizationData (IV-2460)
SSpLocationData	“Sound Types” (IV-2633)
SSpVirtualSourceData	“Sound Types” (IV-2633)
'ssrc'	atom 'ssrc' (IV-2586)
StandardFileReply	struct StandardFileReply (IV-2461)
StateBlock	struct StateBlock (IV-2463)
StateBlockPtr	“Sound Types” (IV-2633)
'stbl'	atom 'stbl' (IV-2587)
'stco'	atom 'stco' (IV-2589)
StdPixProcPtr	callback StdPixProc (III-2149)
StdPixUPP	“Universal Procedure Pointers” (IV-2641)
STHandle	“Text Types” (IV-2640)
StrFileName	“String Types” (IV-2637)
StringHandle	“String Types” (IV-2637)
StringPtr	“String Types” (IV-2637)
StringRangeRecord	struct StringRangeRecord (IV-2463)
'strt'	atom 'strt' (IV-2589)
'stsc'	atom 'stsc' (IV-2590)
'stsd'	atom 'stsd' (IV-2591)
'stsh'	atom 'stsh' (IV-2592)
'stss'	atom 'stss' (IV-2593)
'stsz'	atom 'stsz' (IV-2594)
'stts'	atom 'stts' (IV-2595)
Style	“Graphics Types” (IV-2624)
StyleField	“Graphics Types” (IV-2624)
StyleParameter	“Graphics Types” (IV-2624)
StyleRun	“Text Types” (IV-2640)
'sync'	atom 'sync' (IV-2596)
SynthesizerConnections	struct SynthesizerConnections (IV-2464)
SynthesizerDescription	struct SynthesizerDescription (IV-2465)
'tbox'	atom 'tbox' (IV-2597)
'tcmi'	atom 'tcmi' (IV-2597)
TCTextOptions	struct TCTextOptions (IV-2468)
TCTextOptionsPtr	“Text Types” (IV-2640)
TEClickLoopUPP	“Universal Procedure Pointers” (IV-2641)
TEHandle	“Text Types” (IV-2640)
TEPtr	“Text Types” (IV-2640)
TERec	struct TERec (IV-2469)

TEStyleRec	struct TEStyleRec (IV-2473)
TextDescription	struct TextDescription (IV-2474)
TextDescriptionHandle	“Text Types” (IV-2640)
TextDescriptionPtr	“Text Types” (IV-2640)
TextDisplayData	struct TextDisplayData (IV-2476)
TextExportComponent	“Component Types” (IV-2621)
TextMediaProcPtr	callback TextMediaProc (III-2150)
TextMediaUPP	“Universal Procedure Pointers” (IV-2641)
ThreeDeeDescription	struct ThreeDeeDescription (IV-2478)
ThreeDeeDescriptionHandle	“Media Handling Types” (IV-2626)
ThreeDeeDescriptionPtr	“Media Handling Types” (IV-2626)
ThreeDeeNonLinearSample	struct ThreeDeeNonLinearSample (IV-2479)
ThreeDeeNonLinearTweenHeaderAtom	struct ThreeDeeNonLinearTweenHeaderAtom (IV-2480)
ThreeDeeVRObjectSample	struct ThreeDeeVRObjectSample (IV-2480)
TimeBase	“Timing and Callback Types” (IV-2640)
TimeBaseFlags	“Timing and Callback Types” (IV-2640)
TimeBaseRecord	struct TimeBaseRecord (IV-2481)
TimeBaseStatus	“Timing and Callback Types” (IV-2640)
TimeCodeCounter	struct TimeCodeCounter (IV-2481)
TimeCodeDef	struct TimeCodeDef (IV-2482)
TimeCodeDescription	struct TimeCodeDescription (IV-2483)
TimeCodeDescriptionHandle	“Timing and Callback Types” (IV-2640)
TimeCodeDescriptionPtr	“Timing and Callback Types” (IV-2640)
TimeCodeRecord	struct TimeCodeRecord (IV-2484)
TimeCodeTime	struct TimeCodeTime (IV-2485)
TimeRecord	struct TimeRecord (IV-2486)
TimeScale	“Timing and Callback Types” (IV-2640)
TimeToSampleNum	struct TimeToSampleNum (IV-2486)
TimeValue	“Timing and Callback Types” (IV-2640)
TimeValue64	“Timing and Callback Types” (IV-2640)
'tkhd'	atom 'tkhd' (IV-2598)
'tmax'	atom 'tmax' (IV-2598)
'tmcd'	atom 'tmcd' (IV-2599)
'tmin'	atom 'tmin' (IV-2599)
ToneDescription	struct ToneDescription (IV-2487)
'tpyl'	atom 'tpyl' (IV-2600)
TQ3Matrix4x4	struct TQ3Matrix4x4 (IV-2488)
Track	“Movie Types” (IV-2628)
TrackDirectory	“Movie Types” (IV-2628)
TrackDirectoryEntry	struct TrackDirectoryEntry (IV-2489)
TrackEditState	“Movie Types” (IV-2628)
TrackEditStateRecord	struct TrackEditStateRecord (IV-2489)
TrackHeader	struct TrackHeader (IV-2489)
TrackLoadSettings	struct TrackLoadSettings (IV-2491)
TrackRecord	struct TrackRecord (IV-2492)
TrackTransferProcPtr	callback TrackTransferProc (III-2151)
TrackTransferUPP	“Universal Procedure Pointers” (IV-2641)
'trak'	atom 'trak' (IV-2600)

'tref'	atom 'tref' (IV-2602)
'trpy'	atom 'trpy' (IV-2603)
TuneCallbackProcPtr	callback TuneCallbackProc (III-2152)
TuneCallbackUPP	"Universal Procedure Pointers" (IV-2641)
TunePlayCallbackProcPtr	callback TunePlayCallbackProc (III-2152)
TunePlayCallbackUPP	"Universal Procedure Pointers" (IV-2641)
TunePlayer	"Music Types" (IV-2630)
TuneStatus	struct TuneStatus (IV-2492)
'twdt'	atom 'twdt' (IV-2604)
'twdu'	atom 'twdu' (IV-2604)
TweenerComponent	"Component Types" (IV-2621)
TweenerDataProcPtr	callback TweenerDataProc (III-2153)
TweenerDataUPP	"Universal Procedure Pointers" (IV-2641)
TweenRecord	struct TweenRecord (IV-2493)
TweenSequenceEntryRecord	struct TweenSequenceEntryRecord (IV-2494)
TweenV1Record	struct TweenV1Record (IV-2495)
'twen'	atom 'twen' (IV-2605)
'twnt'	atom 'twnt' (IV-2605)
'twst'	atom 'twst' (IV-2606)
' ty'	atom ' ty' (IV-2606)
'udta'	atom 'udta' (IV-2607)
UInt16	"Numeric Types" (IV-2631)
UInt32	"Numeric Types" (IV-2631)
UInt64	"Numeric Types" (IV-2631)
UInt8	"Numeric Types" (IV-2631)
UniChar	"String Types" (IV-2637)
UniCharCount	"String Types" (IV-2637)
UniversalProcHandle	"Universal Procedure Pointers" (IV-2641)
UniversalProcPtr	"Universal Procedure Pointers" (IV-2641)
UnsignedFixed	"Numeric Types" (IV-2631)
UnsignedFixedPtr	"Numeric Types" (IV-2631)
UnsignedWide	struct UnsignedWide (IV-2496)
UnsignedWidePtr	"Numeric Types" (IV-2631)
UserData	"Movie Types" (IV-2628)
UserDataRecord	struct UserDataRecord (IV-2496)
VDCompressionList	struct VDCompressionList (IV-2497)
VDCompressionListHandle	"Video Types" (IV-2646)
VDCompressionListPtr	"Video Types" (IV-2646)
VDGammaRecord	struct VGammaRecord (IV-2498)
VDGamRecPtr	"Graphics Types" (IV-2624)
VdigBufferRec	struct VdigBufferRec (IV-2498)
VdigBufferRecList	struct VdigBufferRecList (IV-2499)
VdigBufferRecListHandle	"Video Types" (IV-2646)
VdigBufferRecListPtr	"Video Types" (IV-2646)
VdigIntProcPtr	callback VdigIntProc (III-2154)
VdigIntUPP	"Universal Procedure Pointers" (IV-2641)
VdigType	struct VdigType (IV-2500)
VdigTypeList	struct VdigTypeList (IV-2501)
VHSelect	"Numeric Types" (IV-2631)

'vide'	atom 'vide' (IV-2610)
VideoBottles	struct VideoBottles (IV-2501)
VideoDigitizerComponent	"Video Types" (IV-2646)
VideoDigitizerError	"Video Types" (IV-2646)
VideoMediaInfoHeader	struct VideoMediaInfoHeader (IV-2503)
'vmhd'(media)	atom 'vmhd'(media) (IV-2611)
'vmhd'(transfer)	atom 'vmhd'(transfer) (IV-2612)
'vrcp'	atom 'vrcp' (IV-2612)
'vrni'	atom 'vrni' (IV-2613)
'vrnp'	atom 'vrnp' (IV-2614)
'vrsc'	atom 'vrsc' (IV-2614)
'wide'	atom 'wide' (IV-2614)
WidePtr	"Numeric Types" (IV-2631)
WindowPtr	"Graphics Types" (IV-2624)
WindowRef	"Graphics Types" (IV-2624)
'WLOC'	atom 'WLOC' (IV-2615)
WordBreakUPP	"Universal Procedure Pointers" (IV-2641)
'wtxt'	atom 'wtxt' (IV-2616)
XMLAttribute	struct XMLAttribute (IV-2504)
XMLAttributePtr	"SMIL Types" (IV-2633)
XMLAttributeValue	"SMIL Types" (IV-2633)
XMLContent	struct XMLContent (IV-2505)
XMLContentPtr	"SMIL Types" (IV-2633)
XMLDoc	"SMIL Types" (IV-2633)
XMLDocRecord	struct XMLDocRecord (IV-2505)
XMLElement	struct XMLElement (IV-2505)
XMLElementContent	"SMIL Types" (IV-2633)
XMLElementPtr	"SMIL Types" (IV-2633)

Index of Constants

0x0000	function DragAlignedGrayRgn (I-304)
0x00000001	function GetMediaQuality (I-441)
0x00000002	function ImageCodecNewMemory (II-729)
0x00000004	struct ParameterAtomTypeAndID (IV-2327)
0x00000010	function GetMediaQuality (I-441)
0x00000100	function GetMediaQuality (I-441)
0x00001000	function GetMediaQuality (I-441)
0x0001	function CDSequenceNewMemory (I-84)
0x00010000	function GetMediaQuality (I-441)
0x0002	function CDSequenceNewMemory (I-84)
0x00100000	function GetMediaQuality (I-441)
AAPNotCreatedErr	“Error Codes” (IV-2663)
AAPNotFoundErr	“Error Codes” (IV-2663)
activateEvt	struct EventRecord (IV-2246)
activeFlag	function QTVRMouseDown (II-1391)
adbAddrMask	struct EventRecord (IV-2246)
addMax	“Graphics Transfer Modes” (IV-2670)
addOver	function SetDSequenceTransferMode (III-1591)
addPin	function SetDSequenceTransferMode (III-1591)
adMax	function SetDSequenceTransferMode (III-1591)
adMin	function SetDSequenceTransferMode (III-1591)
alignPix	function NewImageGWorld (II-1066)
'AllF'	atom 'AllF' (IV-2511)
allInit	struct GDevice (IV-2264)
alphaLock	function QTVRMouseDown (II-1391)
alphaStage	struct NumVersion (IV-2324)
anyCodec	function CompressSequenceBegin (I-119)
AppleDataCompressorSubType	atom 'dcom' (IV-2527)
ASDBadForkErr	“Error Codes” (IV-2663)
ASDBadHeaderErr	“Error Codes” (IV-2663)
ASDEntryNotFoundErr	“Error Codes” (IV-2663)
atomIndexInvalidErr	“Error Codes” (IV-2663)
atomsNotOfSameTypeErr	“Error Codes” (IV-2663)
AudioMediaCharacteristic	function GetMovieNextInterestingTime (I-480)
autoKey	struct EventRecord (IV-2246)
auxiliaryExportDataUnavailable	“Error Codes” (IV-2663)
availableCmd	“Sound Commands” (IV-2691)
'avr '	atom 'wtxt' (IV-2616)
badCallOrderErr	atom 'wtxt' (IV-2616)
badComponentSelector	function ComponentSetTarget (I-112)

badComponentType	"Error Codes" (IV-2663)
badDataRefIndex	"Error Codes" (IV-2663)
badDepthErr	atom 'wtxt' (IV-2616)
badEditIndex	"Error Codes" (IV-2663)
badEditList	"Error Codes" (IV-2663)
badImageDescription	"Error Codes" (IV-2663)
badPublicMovieAtom	"Error Codes" (IV-2663)
badTrackIndex	"Error Codes" (IV-2663)
BandwidthManagementPrefsType	function GetQuickTimePreference (I-507)
BaseMediaType	"Component Identifiers" (IV-2657)
bestCompressionCodec	function CompressSequenceBegin (I-119)
bestFidelityCodec	function CompressSequenceBegin (I-119)
bestSpeedCodec	function CompressSequenceBegin (I-119)
betaStage	struct NumVersion (IV-2324)
blackColor	"Color Constants" (IV-2657)
blend	function SetDSequenceTransferMode (III-1591)
blueColor	"Color Constants" (IV-2657)
btnState	function QTVRMouseDown (II-1391)
bufferCmd	"Sound Commands" (IV-2691)
bufferLength	function SPBRecord (III-1878)
bufferPtr	function SPBRecord (III-1878)
burstDevice	struct GDevice (IV-2264)
calArabicCivil	"Localization Codes" (IV-2673)
calArabicLunar	"Localization Codes" (IV-2673)
calCoptic	"Localization Codes" (IV-2673)
calGregorian	"Localization Codes" (IV-2673)
calJapanese	"Localization Codes" (IV-2673)
calJewish	"Localization Codes" (IV-2673)
callBackAtExtremes	function NewCallBack (II-1053)
callBackAtInterrupt	function ClockNewCallBack (I-96)
callBackAtRate	function ClockNewCallBack (I-96)
callBackAtTime	function ClockNewCallBack (I-96)
callBackAtTimeJump	function ClockNewCallBack (I-96)
callBackCmd	"Sound Commands" (IV-2691)
callNotSupportedByNodeErr	"Error Codes" (IV-2663)
callOldBits	function StdPix (III-1912)
callStdBits	function StdPix (III-1912)
calPersian	"Localization Codes" (IV-2673)
canMovieExportAuxDataHandle	struct ComponentDescription (IV-2212)
canMovieExportFiles	struct ComponentDescription (IV-2212)
canMovieExportFromProcedures	struct ComponentDescription (IV-2212)
canMovieExportHandles	struct ComponentDescription (IV-2212)
canMovieExportValidateMovie	struct ComponentDescription (IV-2212)
canMovieImportDataReferences	struct ComponentDescription (IV-2212)
canMovieImportFiles	struct ComponentDescription (IV-2212)
canMovieImportHandles	struct ComponentDescription (IV-2212)
canMovieImportInPlace	struct ComponentDescription (IV-2212)
canMovieImportPartial	struct ComponentDescription (IV-2212)
canMovieImportValidateDataReferences	struct ComponentDescription (IV-2212)

canMovieImportValidateFile	struct ComponentDescription (IV-2212)
canMovieImportValidateHandles	struct ComponentDescription (IV-2212)
cannotBeLeafAtomErr	“Error Codes” (IV-2663)
cannotFindAtomErr	“Error Codes” (IV-2663)
cantCreateSingleForkFile	atom 'wtxt' (IV-2616)
cantEnableTrack	“Error Codes” (IV-2663)
cantFindHandler	“Error Codes” (IV-2663)
cantOpenHandler	“Error Codes” (IV-2663)
cantPutPublicMovieAtom	“Error Codes” (IV-2663)
cantReceiveFromSynthesizerOSErr	“Error Codes” (IV-2663)
cantSendToSynthesizerOSErr	“Error Codes” (IV-2663)
channelFlagDontOpenResFile	struct ComponentDescription (IV-2212)
channelFlagHasDependency	struct ComponentDescription (IV-2212)
channelPlayAllData	function SGGetChannelPlayFlags (III-1705)
channelPlayFast	function SGGetChannelPlayFlags (III-1705)
channelPlayHighQuality	function SGGetChannelPlayFlags (III-1705)
channelPlayNormal	function SGGetChannelPlayFlags (III-1705)
'chap'	atom 'chap' (IV-2512)
charCodeMask	struct EventRecord (IV-2246)
'clip'	atom 'clip' (IV-2513)
ClipAID	“Atom ID Codes” (IV-2649)
clipPix	function NewImageGWorld (II-1066)
clockComponentCmd	“Sound Commands” (IV-2691)
clockComponentType	“Component Identifiers” (IV-2657)
clutType	struct GDevice (IV-2264)
cmdKey	function QTVRMouseDown (II-1391)
'cmov'	atom 'cmov' (IV-2514)
cmpSH	struct CmpSoundHeader (IV-2176)
'cmvd'	atom 'cmvd' (IV-2514)
'co64'	atom 'co64' (IV-2515)
'CODE'	struct ResourceSpec (IV-2402)
codecAbortErr	“Error Codes” (IV-2663)
codecBadDataErr	“Error Codes” (IV-2663)
codecCanAsync	struct CodecCapabilities (IV-2179)
codecCanAsyncWhen	struct CodecCapabilities (IV-2179)
codecCanClipRectangular	struct CodecCapabilities (IV-2179)
codecCanClipVertical	struct CodecCapabilities (IV-2179)
codecCanCopyPrev	struct CodecCapabilities (IV-2179)
codecCanCopyPrevComp	struct CodecCapabilities (IV-2179)
codecCanFastDither	struct CodecCapabilities (IV-2179)
codecCanMakeMask	struct CodecCapabilities (IV-2179)
codecCanManagePrevBuffer	struct CodecCapabilities (IV-2179)
codecCanMask	struct CodecCapabilities (IV-2179)
codecCanMatte	struct CodecCapabilities (IV-2179)
codecCanRemapColor	struct CodecCapabilities (IV-2179)
codecCanScale	struct CodecCapabilities (IV-2179)
codecCanShieldCursor	struct CodecCapabilities (IV-2179)
codecCanShift	struct CodecCapabilities (IV-2179)
codecCanSpool	struct CodecCapabilities (IV-2179)

codecCanSrcExtract	struct CodecCapabilities (IV-2179)
codecCantQueueErr	"Error Codes" (IV-2663)
codecCanTransferMode	struct CodecCapabilities (IV-2179)
codecCanTransform	struct CodecCapabilities (IV-2179)
codecCantWhenErr	"Error Codes" (IV-2663)
codecCompletionDest	function ICMDecompressComplete (II-671)
codecCompletionDontUnshield	function ICMDecompressComplete (II-671)
codecCompletionSource	function ICMDecompressComplete (II-671)
codecConditionCatchUpDiff	struct CodecDecompressParams (IV-2190)
codecConditionCodecChangedMask	struct CodecCompressParams (IV-2184)
codecConditionDoCursor	struct CodecDecompressParams (IV-2190)
codecConditionErr	"Error Codes" (IV-2663)
codecConditionFirstBand	struct CodecCompressParams (IV-2184)
codecConditionFirstFrame	struct CodecDecompressParams (IV-2190)
codecConditionFirstScreen	struct CodecDecompressParams (IV-2190)
codecConditionLastBand	struct CodecCompressParams (IV-2184)
codecConditionMaskMayBeChanged	struct CodecDecompressParams (IV-2190)
codecConditionNewAccuracy	struct CodecDecompressParams (IV-2190)
codecConditionNewClut	struct CodecDecompressParams (IV-2190)
codecConditionNewDepth	struct CodecDecompressParams (IV-2190)
codecConditionNewDestination	struct CodecDecompressParams (IV-2190)
codecConditionNewMatte	struct CodecDecompressParams (IV-2190)
codecConditionNewSrcRect	struct CodecDecompressParams (IV-2190)
codecConditionNewTransferMode	struct CodecDecompressParams (IV-2190)
codecConditionNewTransform	struct CodecDecompressParams (IV-2190)
codecConditionToBuffer	struct CodecDecompressParams (IV-2190)
codecDataVersErr	"Error Codes" (IV-2663)
codecDisabledErr	"Error Codes" (IV-2663)
codecDroppedFrameErr	"Error Codes" (IV-2663)
codecErr	"Error Codes" (IV-2663)
codecExtensionNotFoundErr	"Error Codes" (IV-2663)
codecFlagCatchUpDiff	function DecompressSequenceFrames (I-243)
codecFlagDontOffscreen	function DecompressSequenceFrames (I-243)
codecFlagDontUseImageBuffer	function DecompressSequenceBeginS (I-239)
codecFlagDontUseNewImageBuffer	function DecompressSequenceFrames (I-243)
codecFlagForceKeyFrame	function CompressSequenceFrame (I-124)
codecFlagInterlaceUpdate	function DecompressSequenceFrames (I-243)
codecFlagLiveGrab	function CompressSequenceFrame (I-124)
codecFlagNoScreenUpdate	function DecompressSequenceFrames (I-243)
codecFlagOnlyScreenUpdate	function DecompressSequenceFrames (I-243)
codecFlagUpdatePrevious	function CompressSequenceBegin (I-119)
codecFlagUpdatePreviousComp	function CompressSequenceBegin (I-119)
codecFlagUsedImageBuffer	function DecompressSequenceFrames (I-243)
codecFlagUsedNewImageBuffer	function DecompressSequenceFrames (I-243)
codecFlagUseImageBuffer	function DecompressSequenceBeginS (I-239)
codecFlagUseScreenBuffer	function DecompressSequenceBeginS (I-239)
codecFlagWasCompressed	function CompressSequenceBegin (I-119)
codecHasVolatileBuffer	struct CodecCapabilities (IV-2179)
codecHighQuality	function CompressImage (I-113)

codecOffscreenFailedErr	"Error Codes" (IV-2663)
codecOffscreenFailedPleaseRetryErr	"Error Codes" (IV-2663)
codecOpenErr	"Error Codes" (IV-2663)
codecParameterDialogConfirm	
	function QTIsStandardParameterDialogEvent (II-1186)
codecProgressClose	callback ICMProgressProc (III-2093)
codecProgressOpen	callback ICMProgressProc (III-2093)
codecProgressUpdatePercent	callback ICMProgressProc (III-2093)
codecScreenBufErr	"Error Codes" (IV-2663)
codecSizeErr	"Error Codes" (IV-2663)
codecSpoolErr	"Error Codes" (IV-2663)
codecUnimpErr	"Error Codes" (IV-2663)
codecWantsDestinationPixels	struct CodecCapabilities (IV-2179)
codecWantsSpecialScaling	struct CodecCapabilities (IV-2179)
codecWouldOffscreenErr	"Error Codes" (IV-2663)
ColorTableAID	"Atom ID Codes" (IV-2649)
completionRoutine	function SPBRecord (III-1878)
componentAutoVersionIncludeFlags	
	struct ComponentResourceExtension (IV-2221)
componentDllEntryNotFoundErr	atom 'wtxt' (IV-2616)
componentDllLoadErr	atom 'wtxt' (IV-2616)
componentDoAutoVersion	struct ComponentResourceExtension (IV-2221)
componentHasMultiplePlatforms	struct ComponentResourceExtension (IV-2221)
componentLoadResident	struct ComponentResourceExtension (IV-2221)
componentWantsUnregister	struct ComponentResourceExtension (IV-2221)
compositeIn	function VDGetInputFormat (III-2005)
CompressedMovieAID	"Atom ID Codes" (IV-2649)
CompressedMovieDataAID	"Atom ID Codes" (IV-2649)
compressorComponentType	"Component Identifiers" (IV-2657)
ConnectionSpeedPrefsType	function GetQuickTimePreference (I-507)
constraintReachedErr	"Error Codes" (IV-2663)
controlKey	function QTVRMouseDown (II-1391)
convertClipboardFlag	struct EventRecord (IV-2246)
'@cpy'	atom '@cpy' (IV-2515)
'@day'	atom '@day' (IV-2516)
'@dir'	atom '@dir' (IV-2517)
'@ed1'	atom '@ed1' (IV-2517)
'@fmt'	atom '@fmt' (IV-2518)
'@inf'	atom '@inf' (IV-2519)
'@prd'	atom '@prd' (IV-2519)
'@prf'	atom '@prf' (IV-2520)
'@req'	atom '@req' (IV-2521)
'@src'	atom '@src' (IV-2521)
'@wrt'	atom '@wrt' (IV-2522)
couldNotResolveDataRef	"Error Codes" (IV-2663)
couldNotUseAnExistingSample	"Error Codes" (IV-2663)
count	function SPBRecord (III-1878)
CreateFilePreviewComponentType	"Component Identifiers" (IV-2657)
createMovieFileDeleteCurFile	function ConvertFileToMovieFile (I-129)

createMovieFileDontCreateMovie	function CreateMovieFile (I-145)
createMovieFileDontCreateResFile	function CreateMovieFile (I-145)
createMovieFileDontOpenFile	function CreateMovieFile (I-145)
'crgn'	atom 'crgn' (IV-2523)
'crsr'	atom 'crsr' (IV-2524)
'cspd'	atom 'cspd' (IV-2524)
'ctab'	atom 'ctab' (IV-2525)
'cufa'	atom 'cufa' (IV-2525)
'CURS'	atom 'CURS' (IV-2526)
'cuvw'	atom 'cuvw' (IV-2526)
'cvid'	function SGGetVideoCompressorType (III-1744)
cyanColor	"Color Constants" (IV-2657)
'dasz'	atom 'dasz' (IV-2527)
dataAlreadyClosed	"Error Codes" (IV-2663)
dataAlreadyOpenForWrite	"Error Codes" (IV-2663)
DataCompressionAtomAID	"Atom ID Codes" (IV-2649)
DataHandlerType	"Component Identifiers" (IV-2657)
DataInfoAID	"Atom ID Codes" (IV-2649)
dataNoDataRef	"Error Codes" (IV-2663)
dataNotOpenForRead	"Error Codes" (IV-2663)
dataNotOpenForWrite	"Error Codes" (IV-2663)
DataRefAID	"Atom ID Codes" (IV-2649)
DataRefContainerAID	"Atom ID Codes" (IV-2649)
dataRefSelfReference	function GetMediaDataRef (I-427)
dataRefWasNotResolved	function GetMediaDataRef (I-427)
dbBufferReady	struct SndDoubleBuffer (IV-2442)
dbLastBuffer	struct SndDoubleBuffer (IV-2442)
'dcom'	atom 'dcom' (IV-2527)
decompressorComponentType	"Component Identifiers" (IV-2657)
defaultComponentAnyFlags	function SetDefaultComponent (III-1581)
defaultComponentAnyManufacturer	function SetDefaultComponent (III-1581)
defaultComponentAnySubType	function SetDefaultComponent (III-1581)
defaultComponentIdentical	function SetDefaultComponent (III-1581)
defaultDither	function DrawTrimmedPicture (I-311)
'defi'	atom 'defi' (IV-2528)
'desc'	atom 'desc' (IV-2528)
developStage	struct NumVersion (IV-2324)
dfAntiAlias	function TextMediaAddTESample (III-1933)
dfClipToTextBox	function TextMediaAddTESample (III-1933)
dfContinuousKaraoke	function TextMediaAddTESample (III-1933)
dfContinuousScroll	function TextMediaAddTESample (III-1933)
dfDontAutoScale	function TextMediaAddTESample (III-1933)
dfDontDisplay	function TextMediaAddTESample (III-1933)
dfDropShadow	function TextMediaAddTESample (III-1933)
dfFlowHoriz	function TextMediaAddTESample (III-1933)
dfHorizScroll	function TextMediaAddTESample (III-1933)
dfInverseHilite	function TextMediaAddTESample (III-1933)
dfKeyedText	function TextMediaAddTESample (III-1933)
'dflt'	atom 'dflt' (IV-2529)

dfReverseScroll	function	TextMediaAddTESample	(III-1933)
dfScrollIn	function	TextMediaAddTESample	(III-1933)
dfScrollOut	function	TextMediaAddTESample	(III-1933)
dfShrinkTextBoxToFit	function	TextMediaAddTESample	(III-1933)
dfTextColorHilite	function	TextMediaAddTESample	(III-1933)
digiInDoesBW	struct	DigitizerInfo	(IV-2234)
digiInDoesColor	struct	DigitizerInfo	(IV-2234)
digiInDoesComponent	struct	DigitizerInfo	(IV-2234)
digiInDoesComposite	struct	DigitizerInfo	(IV-2234)
digiInDoesGenLock	struct	DigitizerInfo	(IV-2234)
digiInDoesNTSC	struct	DigitizerInfo	(IV-2234)
digiInDoesPAL	struct	DigitizerInfo	(IV-2234)
digiInDoesSECAM	struct	DigitizerInfo	(IV-2234)
digiInDoesSVideo	“Video Digitizer Capabilities”		(IV-2696)
digiInSignalLock	struct	DigitizerInfo	(IV-2234)
digiInVTR_Broadcast	struct	DigitizerInfo	(IV-2234)
digiOutDoes1	struct	DigitizerInfo	(IV-2234)
digiOutDoes16	struct	DigitizerInfo	(IV-2234)
digiOutDoes2	struct	DigitizerInfo	(IV-2234)
digiOutDoes32	struct	DigitizerInfo	(IV-2234)
digiOutDoes4	struct	DigitizerInfo	(IV-2234)
digiOutDoes8	struct	DigitizerInfo	(IV-2234)
digiOutDoesAsyncGrabs	struct	DigitizerInfo	(IV-2234)
digiOutDoesBlend	struct	DigitizerInfo	(IV-2234)
digiOutDoesCompress	struct	DigitizerInfo	(IV-2234)
digiOutDoesCompressOnly	struct	DigitizerInfo	(IV-2234)
digiOutDoesCompressPartiallyVisible	“Video Digitizer Capabilities”		(IV-2696)
digiOutDoesDither	struct	DigitizerInfo	(IV-2234)
digiOutDoesDMA	struct	DigitizerInfo	(IV-2234)
digiOutDoesDouble	struct	DigitizerInfo	(IV-2234)
digiOutDoesHorizFlip	struct	DigitizerInfo	(IV-2234)
digiOutDoesHW_DMA	“Video Digitizer Capabilities”		(IV-2696)
digiOutDoesHWPlayThru	struct	DigitizerInfo	(IV-2234)
digiOutDoesILUT	struct	DigitizerInfo	(IV-2234)
digiOutDoesKeyColor	struct	DigitizerInfo	(IV-2234)
digiOutDoesMask	struct	DigitizerInfo	(IV-2234)
digiOutDoesPlayThruDuringCompress	struct	DigitizerInfo	(IV-2234)
digiOutDoesQuad	struct	DigitizerInfo	(IV-2234)
digiOutDoesQuarter	struct	DigitizerInfo	(IV-2234)
digiOutDoesRotate	struct	DigitizerInfo	(IV-2234)
digiOutDoesShrink	struct	DigitizerInfo	(IV-2234)
digiOutDoesSixteenth	struct	DigitizerInfo	(IV-2234)
digiOutDoesSkew	struct	DigitizerInfo	(IV-2234)
digiOutDoesStretch	struct	DigitizerInfo	(IV-2234)
digiOutDoesUnreadableScreenBits	struct	DigitizerInfo	(IV-2234)
digiOutDoesVertFlip	struct	DigitizerInfo	(IV-2234)
digiOutDoesWarp	struct	DigitizerInfo	(IV-2234)
digitInDoesSVideo	struct	DigitizerInfo	(IV-2234)

digitizerOff	function VDSetsPlayThruOnOff (III-2050)
digitizerOn	function VDSetsPlayThruOnOff (III-2050)
digiUnimpErr	atom 'wtxt' (IV-2616)
'dimm'	atom 'dimm' (IV-2529)
'dinf'	atom 'dinf' (IV-2530)
directType	struct GDevice (IV-2264)
directXObjectAlreadyExists	"Error Codes" (IV-2663)
diskEvt	struct EventRecord (IV-2246)
ditherCopy	function SetDSequenceTransferMode (III-1591)
ditherPix	function NewImageGWorld (II-1066)
'dmax'	atom 'dmax' (IV-2531)
'dmed'	atom 'dmed' (IV-2531)
dontAutoFileMovieImport	struct ComponentDescription (IV-2212)
dontRegisterWithEasyOpen	struct ComponentDescription (IV-2212)
'dref'	atom 'dref' (IV-2532)
'drep'	atom 'drep' (IV-2532)
duplicateAtomTypeAndIDErr	"Error Codes" (IV-2663)
'eat '	atom 'wtxt' (IV-2616)
EditListAID	"Atom ID Codes" (IV-2649)
EditsAID	"Atom ID Codes" (IV-2649)
'edts'	atom 'edts' (IV-2533)
effectIsRealtime	function QTGetEffectSpeed (II-1176)
elOptionsIncludeNoneInList	function QTGetEffectsList (II-1174)
'elst'	atom 'elst' (IV-2533)
emptyPathErr	"Error Codes" (IV-2663)
'end '	atom 'end ' (IV-2534)
'enda'	atom 'enda' (IV-2535)
endOfDataReached	"Error Codes" (IV-2663)
error	function SPBRecord (III-1878)
evenField1ToEvenFieldOut	function ImageCodecExtractAndCombineFields (II-705)
evenField1ToOddFieldOut	function ImageCodecExtractAndCombineFields (II-705)
evenField2ToEvenFieldOut	function ImageCodecExtractAndCombineFields (II-705)
evenField2ToOddFieldOut	function ImageCodecExtractAndCombineFields (II-705)
'expo'	atom 'expo' (IV-2535)
'ext '	atom 'ext ' (IV-2536)
ext32Device	struct GDevice (IV-2264)
extSH	struct CmpSoundHeader (IV-2176)
featureUnsupported	function QTIsStandardParameterDialogEvent (II-1186)
fileOffsetTooBigErr	"Error Codes" (IV-2663)
finalStage	struct NumVersion (IV-2324)
findTextCaseSensitive	function TextMediaFindNextText (III-1941)
findTextEdgeOK	function TextMediaFindNextText (III-1941)
findTextReverseSearch	function TextMediaFindNextText (III-1941)
findTextUseOffset	function TextMediaFindNextText (III-1941)
findTextWrapAround	function TextMediaFindNextText (III-1941)

fixedCompression	function GetCompressionInfo (I-398)
fixedType	struct GDevice (IV-2264)
'flap'	atom 'flap' (IV-2537)
FlashMediaType	"Component Identifiers" (IV-2657)
flattenActiveTracksOnly	function FlattenMovie (I-372)
flattenAddMovieToDataFork	function FlattenMovie (I-372)
flattenDontInterleaveFlatten	function FlattenMovie (I-372)
flushCmd	"Sound Commands" (IV-2691)
flushFromRam	function LoadMediaIntoRam (II-779)
forceDither	function DrawTrimmedPicture (I-311)
'free'	atom 'free' (IV-2538)
FreeAtomType	"Atom ID Codes" (IV-2649)
'frma'	atom 'frma' (IV-2538)
fsCurPerm	function OpenMovieFile (II-1133)
fsRdPerm	function OpenMovieFile (II-1133)
fsRdWrPerm	function OpenMovieFile (II-1133)
fsRdWrShPerm	function OpenMovieFile (II-1133)
fsWrPerm	function OpenMovieFile (II-1133)
ftAdobePremiereMovie	"File Types and Creators" (IV-2668)
ftAfterDarkModule	"File Types and Creators" (IV-2668)
ftClip3Dgraphic	"File Types and Creators" (IV-2668)
ftCricketChart	"File Types and Creators" (IV-2668)
ftCricketDrawing	"File Types and Creators" (IV-2668)
ftDesignCADDrawing	"File Types and Creators" (IV-2668)
ftImageStudioGraphic	"File Types and Creators" (IV-2668)
ftKaleidaGraphGraphic	"File Types and Creators" (IV-2668)
ftMacFlowChart	"File Types and Creators" (IV-2668)
ftMacSpinDataSet	"File Types and Creators" (IV-2668)
ftMoviePlayerMovie	"File Types and Creators" (IV-2668)
ftPixelPaint	"File Types and Creators" (IV-2668)
ftSuper3DDrawing	"File Types and Creators" (IV-2668)
ftSwivel3DDrawing	"File Types and Creators" (IV-2668)
ftVersaCADDrawing	"File Types and Creators" (IV-2668)
'ftyp'	atom 'ftyp' (IV-2539)
fullScreenAllowEvents	function BeginFullScreen (I-52)
fullScreenDontChangeMenuBar	function BeginFullScreen (I-52)
fullScreenHideCursor	function BeginFullScreen (I-52)
fullScreenPreflightSize	function BeginFullScreen (I-52)
gdDevType	struct GDevice (IV-2264)
'geff'	struct EffectSource (IV-2243)
GenericMediaInfoAID	"Atom ID Codes" (IV-2649)
GenericMediaInfoHeaderAID	"Atom ID Codes" (IV-2649)
getClockComponentCmd	"Sound Commands" (IV-2691)
getRateMultiplierCmd	"Sound Commands" (IV-2691)
getVolumeCmd	"Sound Commands" (IV-2691)
'gmhd'	atom 'gmhd' (IV-2539)
'gmin'	atom 'gmin' (IV-2540)
grabPictCurrentImage	function SGGrabPict (III-1748)
grabPictIgnoreClip	function SGGrabPict (III-1748)

grabPictOffScreen	function SGGrabPict (III-1748)
graphicsImporterDoesntDrawAllPixels	function GraphicsImportDoesDrawAllPixels (I-613)
graphicsImporterDontKnowIfDrawAllPixels	function GraphicsImportDoesDrawAllPixels (I-613)
graphicsImporterDrawsAllPixels	function GraphicsImportDoesDrawAllPixels (I-613)
grayishTextOr	atom 'wtxt' (IV-2616)
greenColor	"Color Constants" (IV-2657)
gwFlagErr	function NewImageGWorld (II-1066)
gWorldsNotSameDepthAndSizeErr	"Error Codes" (IV-2663)
HandleDataHandlerSubType	"Component Identifiers" (IV-2657)
HandlerAID	"Atom ID Codes" (IV-2649)
handlerCanClip	function MediaSetHandlerCapabilities (II-894)
handlerCanMatte	function MediaSetHandlerCapabilities (II-894)
handlerCanTransferMode	function MediaSetHandlerCapabilities (II-894)
handlerCGrafPortOnly	function MediaSetHandlerCapabilities (II-894)
handlerHasSpatial	function MediaSetHandlerCapabilities (II-894)
handlerNeedsBuffer	function MediaSetHandlerCapabilities (II-894)
handlerNoIdle	function MediaSetHandlerCapabilities (II-894)
handlerNoScheduler	function MediaSetHandlerCapabilities (II-894)
handlerWantsTime	function MediaSetHandlerCapabilities (II-894)
hasMovieExportUserInterface	struct ComponentDescription (IV-2212)
hasMovieImportMIMEList	struct ComponentDescription (IV-2212)
hasMovieImportUserInterface	struct ComponentDescription (IV-2212)
'hdlr'	atom 'hdlr' (IV-2540)
hilite	"Graphics Transfer Modes" (IV-2670)
hilite,	atom 'wtxt' (IV-2616)
hilitetransfermode	"Graphics Transfer Modes" (IV-2670)
'hinf'	atom 'hinf' (IV-2541)
'hint'	atom 'hint' (IV-2542)
hintsAllowBlacklining	function SetMediaPlayHints (III-1601)
hintsAllowInterlace	function SetMediaPlayHints (III-1601)
hintsDontPurge	function SetMediaPlayHints (III-1601)
hintsHighQuality	function SetMediaPlayHints (III-1601)
hintsInactive	function SetMediaPlayHints (III-1601)
hintsScrubMode	function SetMediaPlayHints (III-1601)
hintsUseSoundInterp	function SetMediaPlayHints (III-1601)
'hlit'	atom 'hlit' (IV-2543)
'hnti'	atom 'hnti' (IV-2544)
'hots'	atom 'hots' (IV-2544)
'hsin'	atom 'hsin' (IV-2545)
'hspa'	atom 'hspa' (IV-2545)
'htxt'	atom 'htxt' (IV-2546)
icmFrameTimeHasVirtualStartTimeAndDuration	struct ICMFrameTimeRecord (IV-2281)
'ICON'	struct ResourceSpec (IV-2402)
'idat'	atom 'idat' (IV-2546)
identityMatrixType	function GetMatrixType (I-419)

'idsc'	atom 'idsc' (IV-2547)
'iicc'	atom 'iicc' (IV-2547)
illegalChannelOSerr	"Error Codes" (IV-2663)
illegalControllerOSerr	"Error Codes" (IV-2663)
illegalInstrumentOSerr	"Error Codes" (IV-2663)
illegalKnobOSerr	"Error Codes" (IV-2663)
illegalKnobValueOSerr	"Error Codes" (IV-2663)
illegalNoteChannelOSerr	"Error Codes" (IV-2663)
illegalPartOSerr	"Error Codes" (IV-2663)
illegalVoiceAllocationOSerr	"Error Codes" (IV-2663)
'ima4'	function GetCompressionInfo (I-398)
'imag'	atom 'imag' (IV-2547)
'imap'	atom 'imap' (IV-2548)
'imco'	atom 'wtxt' (IV-2616)
'imct'	atom 'imct' (IV-2549)
'imda'	atom 'imda' (IV-2549)
'imdc'	atom 'wtxt' (IV-2616)
'imgp'	atom 'imgp' (IV-2550)
'imgr'	atom 'imgr' (IV-2550)
'impn'	atom 'impn' (IV-2551)
'imre'	atom 'imre' (IV-2552)
'imrg'	atom 'imrg' (IV-2552)
'imrt'	atom 'imrt' (IV-2553)
' in'	atom ' in' (IV-2554)
initPanMask	struct SCStatus (IV-2425)
initSRateMask	struct SCStatus (IV-2425)
initStereoMask	struct SCStatus (IV-2425)
InputMapAID	"Atom ID Codes" (IV-2649)
inRefNum	function SPBRecord (III-1878)
intArabic	"Localization Codes" (IV-2673)
internalComponentErr	"Error Codes" (IV-2663)
internalQuickTimeError	"Error Codes" (IV-2663)
interruptRoutine	function SPBRecord (III-1878)
intEuropean	"Localization Codes" (IV-2673)
intJapanese	"Localization Codes" (IV-2673)
intRoman	"Localization Codes" (IV-2673)
intWestern	"Localization Codes" (IV-2673)
invalidAtomContainerErr	"Error Codes" (IV-2663)
invalidAtomErr	"Error Codes" (IV-2663)
invalidAtomTypeErr	"Error Codes" (IV-2663)
invalidChunkCache	"Error Codes" (IV-2663)
invalidChunkNum	"Error Codes" (IV-2663)
invalidDataRef	"Error Codes" (IV-2663)
invalidDataRefContainer	"Error Codes" (IV-2663)
invalidDuration	"Error Codes" (IV-2663)
invalidEditState	"Error Codes" (IV-2663)
invalidHandler	"Error Codes" (IV-2663)
invalidHotSpotIDErr	"Error Codes" (IV-2663)
invalidImageIndexErr	"Error Codes" (IV-2663)

invalidMedia	"Error Codes" (IV-2663)
invalidMovie	"Error Codes" (IV-2663)
invalidNodeFormatErr	"Error Codes" (IV-2663)
invalidNodeIDErr	"Error Codes" (IV-2663)
invalidRect	"Error Codes" (IV-2663)
invalidSampleDescIndex	"Error Codes" (IV-2663)
invalidSampleDescription	"Error Codes" (IV-2663)
invalidSampleNum	"Error Codes" (IV-2663)
invalidSampleTable	"Error Codes" (IV-2663)
invalidSpriteIDErr	"Error Codes" (IV-2663)
invalidSpriteIndexErr	"Error Codes" (IV-2663)
invalidSpritePropertyErr	"Error Codes" (IV-2663)
invalidSpriteWorldPropertyErr	"Error Codes" (IV-2663)
invalidTime	"Error Codes" (IV-2663)
invalidTrack	"Error Codes" (IV-2663)
invalidViewStateErr	"Error Codes" (IV-2663)
'jpeg'	function SGGetVideoCompressorType (III-1744)
k16BE565PixelFormat	function QTVRGetBackBufferSettings (II-1350)
k16BitBigEndianFormat	"Sound Formats" (IV-2692)
k16BitIn	"Sound Device Features" (IV-2692)
k16BitLittleEndianFormat	"Sound Formats" (IV-2692)
k16BitOut	"Sound Device Features" (IV-2692)
k16GrayCodecType	"Codec Identifiers" (IV-2655)
k16LE5551PixelFormat	"Pixel Formats" (IV-2686)
k16LE555PixelFormat	function QTVRGetBackBufferSettings (II-1350)
k16LE565PixelFormat	function QTVRGetBackBufferSettings (II-1350)
k24BGRPixelFormat	function QTVRGetBackBufferSettings (II-1350)
k24BitFormat	"Sound Formats" (IV-2692)
k32ABGRPixelFormat	function QTVRGetBackBufferSettings (II-1350)
k32AlphaGrayCodecType	"Codec Identifiers" (IV-2655)
k32ARGBPixelFormat	function QTVRGetBackBufferSettings (II-1350)
k32BGRAPixelFormat	function QTVRGetBackBufferSettings (II-1350)
k32BitFormat	"Sound Formats" (IV-2692)
k32RGBAPixelFormat	function QTVRGetBackBufferSettings (II-1350)
k48RGBCodecType	"Codec Identifiers" (IV-2655)
k64ARGBCodecType	"Codec Identifiers" (IV-2655)
k8BitOffsetBinaryFormat	"Sound Formats" (IV-2692)
k8BitRawIn	"Sound Device Features" (IV-2692)
k8BitRawOut	"Sound Device Features" (IV-2692)
k8BitTwosIn	"Sound Device Features" (IV-2692)
k8BitTwosOut	"Sound Device Features" (IV-2692)
kAccessKeySystemFlag	function QTRegisterAccessKey (II-1214)
kAction	"Atom ID Codes" (IV-2649)
kActionFlags	"Atom ID Codes" (IV-2649)
kActionListAtomType	"Atom ID Codes" (IV-2649)
kActionParameter	"Atom ID Codes" (IV-2649)
kActionParameterMaxValue	"Atom ID Codes" (IV-2649)
kActionParameterMinValue	"Atom ID Codes" (IV-2649)
kActionTarget	"Atom ID Codes" (IV-2649)

kALawCompression	"Sound Formats" (IV-2692)
kAlphaCompositorTransitionType	"Effects Codes" (IV-2662)
kAlphaGainImageFilterType	"Effects Codes" (IV-2662)
kAnimationCodecType	"Codec Identifiers" (IV-2655)
kAudioEndianAtomType	"Atom ID Codes" (IV-2649)
kAudioFormatAtomType	"Atom ID Codes" (IV-2649)
kAudioTerminatorAtomType	"Atom ID Codes" (IV-2649)
kAudioVBRAtomType	"Atom ID Codes" (IV-2649)
kAVRJPEGCodecType	"Codec Identifiers" (IV-2655)
kBaseCodecType	"Codec Identifiers" (IV-2655)
kBlurImageFilterType	"Effects Codes" (IV-2662)
kBMPCodecType	"Codec Identifiers" (IV-2655)
kBrightnessContrastImageFilterType	"Effects Codes" (IV-2662)
kCDXA2Compression	"Sound Formats" (IV-2692)
kCDXA4Compression	"Sound Formats" (IV-2692)
kChromaKeyTransitionType	"Effects Codes" (IV-2662)
kCinepakCodecType	"Codec Identifiers" (IV-2655)
kCloudCodecType	"Codec Identifiers" (IV-2655)
kCMYKCodecType	"Codec Identifiers" (IV-2655)
kColorSyncImageFilterType	"Effects Codes" (IV-2662)
kColorTintImageFilterType	"Effects Codes" (IV-2662)
kCommentAtomType	"Atom ID Codes" (IV-2649)
kComponentCanDoSelect	struct ComponentParameters (IV-2216)
kComponentCloseSelect	struct ComponentParameters (IV-2216)
kComponentExecuteWiredActionSelect	struct ComponentParameters (IV-2216)
kComponentGetMPWorkFunctionSelect	struct ComponentParameters (IV-2216)
kComponentGetPublicResourceSelect	struct ComponentParameters (IV-2216)
kComponentOpenSelect	struct ComponentParameters (IV-2216)
kComponentRegisterSelect	struct ComponentParameters (IV-2216)
kComponentTargetSelect	struct ComponentParameters (IV-2216)
kComponentUnregisterSelect	struct ComponentParameters (IV-2216)
kComponentVersionSelect	struct ComponentParameters (IV-2216)
kComponentVideoCodecType	"Codec Identifiers" (IV-2655)
kComponentVideoSigned	atom 'wtxt' (IV-2616)
kComponentVideoUnsigned	atom 'wtxt' (IV-2616)
kConditionalAtomType	"Atom ID Codes" (IV-2649)
kConnectionActive	struct QTSTransportPref (IV-2388)
kConnectionUseSystemPref	struct QTSTransportPref (IV-2388)
kControllerAfterTouch	atom 'wtxt' (IV-2616)
kControllerBalance	atom 'wtxt' (IV-2616)
kControllerBreath	atom 'wtxt' (IV-2616)
kControllerCeleste	atom 'wtxt' (IV-2616)
kControllerChorus	atom 'wtxt' (IV-2616)
kControllerEditPart	atom 'wtxt' (IV-2616)
kControllerExpression	atom 'wtxt' (IV-2616)
kControllerFoot	atom 'wtxt' (IV-2616)
kControllerLever1	atom 'wtxt' (IV-2616)
kControllerLever2	"Music Controllers" (IV-2682)
kControllerLever3	"Music Controllers" (IV-2682)

kControllerLever4	“Music Controllers” (IV-2682)
kControllerLever5	“Music Controllers” (IV-2682)
kControllerLever6	“Music Controllers” (IV-2682)
kControllerLever7	“Music Controllers” (IV-2682)
kControllerLever8	“Music Controllers” (IV-2682)
kControllerMasterTune	atom 'wtxt' (IV-2616)
kControllerModulationWheel	atom 'wtxt' (IV-2616)
kControllerPan	atom 'wtxt' (IV-2616)
kControllerPhaser	atom 'wtxt' (IV-2616)
kControllerPitchBend	atom 'wtxt' (IV-2616)
kControllerPortamentoTime	atom 'wtxt' (IV-2616)
kControllerReverb	atom 'wtxt' (IV-2616)
kControllerSoftPedal	atom 'wtxt' (IV-2616)
kControllerSostenuto	atom 'wtxt' (IV-2616)
kControllerSustain	atom 'wtxt' (IV-2616)
kControllerTremolo	atom 'wtxt' (IV-2616)
kControllerVolume	atom 'wtxt' (IV-2616)
kConvolveImageFilterType	“Effects Codes” (IV-2662)
kCreateSoundSource	“Sound Device Features” (IV-2692)
kCrossFadeTransitionType	“Effects Codes” (IV-2662)
kCustomActionHandler	“Atom ID Codes” (IV-2649)
kCustomHandlerDesc	“Atom ID Codes” (IV-2649)
kCustomHandlerID	“Atom ID Codes” (IV-2649)
kDataHCanRead	function DataHCanUseDataRef (I-175)
kDataHCanStreamingWrite	function DataHCanUseDataRef (I-175)
kDataHCanWrite	function DataHCanUseDataRef (I-175)
kDataHChokeToMovieDataRate	struct DataHChokeAtomRecord (IV-2228)
kDataHChokeToParam	struct DataHChokeAtomRecord (IV-2228)
kDataHExtendedSchedule	struct DataHScheduleRecord (IV-2229)
kDataHMustCheckDataRef	struct DataHVolumelistRecord (IV-2230)
kDataHSpecialRead	function DataHCanUseDataRef (I-175)
kDataHSpecialReadFile	function DataHCanUseDataRef (I-175)
kDataHSpecialWrite	function DataHCanUseDataRef (I-175)
kDataRate144ModemRate	struct QTAltDataRateRecord (IV-2347)
kDataRate288ModemRate	struct QTAltDataRateRecord (IV-2347)
kDataRateDualISDNRate	struct QTAltDataRateRecord (IV-2347)
kDataRateInfiniteRate	struct QTAltDataRateRecord (IV-2347)
kDataRateISDNRate	struct QTAltDataRateRecord (IV-2347)
kDataRateT1Rate	struct QTAltDataRateRecord (IV-2347)
kDataRefIsSelfContained	struct ReferenceMovieDataRefRecord (IV-2400)
kDDSurfaceLocked	function QTSetDDPrimarySurface (II-1226)
kDDSurfaceStatic	function QTSetDDPrimarySurface (II-1226)
kDirectoryPathOnly	function FSSpecToNativePathName (I-379)
kDisableWhenEqual	struct ParameterDataBehavior (IV-2328)
kDisableWhenGreaterThan	struct ParameterDataBehavior (IV-2328)
kDisableWhenLessThan	struct ParameterDataBehavior (IV-2328)
kDisableWhenNotEqual	struct ParameterDataBehavior (IV-2328)
kDontPassSelector	struct RoutineRecord (IV-2405)
kDontUseValidateToFindGraphicsImporter	

```

function GetGraphicsImporterForDataRefWithFlags (I-413)
kDVAudioFormat "Sound Formats" (IV-2692)
kDVCMNTSCCodecType "Codec Identifiers" (IV-2655)
kDVCPALCodecType "Codec Identifiers" (IV-2655)
kDVCPProNTSCCodecType "Codec Identifiers" (IV-2655)
kDVCPProPALCodecType "Codec Identifiers" (IV-2655)
kDVIIntelIMAFormat "Sound Formats" (IV-2692)
kEdgeDetectImageFilterType "Effects Codes" (IV-2662)
keepInRam function LoadMediaIntoRam (II-779)
keepLocal function NewImageGWorld (II-1066)
kEffectGenericType struct EffectSource (IV-2243)
kEffectManufacturerAtom "Atom ID Codes" (IV-2649)
kEffectNameAtom "Atom ID Codes" (IV-2649)
kEffectRawSource struct EffectSource (IV-2243)
kEffectTypeAtom "Atom ID Codes" (IV-2649)
kEmbossImageFilterType "Effects Codes" (IV-2662)
kExplodeTransitionType "Effects Codes" (IV-2662)
kExpressionContainerAtomType "Atom ID Codes" (IV-2649)
kExtendedSoundBufferSizeValid
kExtendedSoundSampleCountNotValid struct ExtendedScheduledSoundHeader (IV-2251)
keyCodeMask struct ExtendedScheduledSoundHeader (IV-2251)
keyDown struct EventRecord (IV-2246)
keyUp struct EventRecord (IV-2246)
kFileNameOnly function FSSpecToNativePathName (I-379)
kFilmNoiseImageFilterType "Effects Codes" (IV-2662)
kFireCodecType "Codec Identifiers" (IV-2655)
kFLCCCodecType "Codec Identifiers" (IV-2655)
kFloat32Format "Sound Formats" (IV-2692)
kFloat64Format "Sound Formats" (IV-2692)
kFragmentNeedsPreparing struct RoutineRecord (IV-2405)
kFullNativePath function FSSpecToNativePathName (I-379)
kFullVolume function GetTrackVolume (I-547)
kGeneralEventAtomicInstrument atom 'wtxt' (IV-2616)
kGeneralEventKnob atom 'wtxt' (IV-2616)
kGeneralEventMIDIChannel atom 'wtxt' (IV-2616)
kGeneralEventNoOp atom 'wtxt' (IV-2616)
kGeneralEventNoteRequest atom 'wtxt' (IV-2616)
kGeneralEventPartChange atom 'wtxt' (IV-2616)
kGeneralEventPartKey "QTMA Events" (IV-2686)
kGeneralEventPartMix atom 'wtxt' (IV-2616)
kGeneralEventTuneDifference atom 'wtxt' (IV-2616)
kGeneralEventUsedNotes atom 'wtxt' (IV-2616)
kGenericMusicBank0...kGenericMusicBank32
function MusicGenericConfigure (II-982)
kGenericMusicCallInstrument function MusicGenericConfigure (II-982)
kGenericMusicCallKnobs function MusicGenericConfigure (II-982)
kGenericMusicCallNumber function MusicGenericConfigure (II-982)

```

kGenericMusicCallParts	function MusicGenericConfigure (II-982)
kGenericMusicCallROMInstrument	function MusicGenericConfigure (II-982)
kGenericMusicDoMIDI	function MusicGenericConfigure (II-982)
kGenericMusicDrumKnob	function MusicDerivedSetKnob (II-976)
kGenericMusicErsatzMIDI	function MusicGenericConfigure (II-982)
kGenericMusicInstrumentKnob	function MusicDerivedSetKnob (II-976)
kGenericMusicKnob	function MusicDerivedSetKnob (II-976)
kGetAtomicInstAllKnobs	function InstrumentGetInst (II-767)
kGetAtomicInstNoExpandedSamples	function InstrumentGetInst (II-767)
kGetAtomicInstNoInstrumentInfo	function InstrumentGetInst (II-767)
kGetAtomicInstNoKnobList	function InstrumentGetInst (II-767)
kGetAtomicInstNoOriginalSamples	function InstrumentGetInst (II-767)
kGetAtomicInstNoSamples	function InstrumentGetInst (II-767)
kGetAtomicInstOriginalKnobList	function InstrumentGetInst (II-767)
kGetInstrumentInfoMidiUserInst	function InstrumentGetInfo (II-766)
kGetInstrumentInfoNoBuiltin	function InstrumentGetInfo (II-766)
kGetInstrumentInfoNoIText	function InstrumentGetInfo (II-766)
kGetMovieImporterDontConsiderGraphicsImporters	function GetMovieImporterForDataRef (I-472)
kGIFCodecType	"Codec Identifiers" (IV-2655)
kGMSynthComponentSubType	function InstrumentGetSynthesizerType (II-768)
kGradientTransitionType	"Effects Codes" (IV-2662)
kGraphicsCodecType	"Codec Identifiers" (IV-2655)
kGraphicsExportDescription	"Atom ID Codes" (IV-2649)
kGraphicsExportExtension	"Atom ID Codes" (IV-2649)
kGraphicsExportFileType	"Atom ID Codes" (IV-2649)
kGraphicsExportGroup	"Atom ID Codes" (IV-2649)
kGraphicsExportMIMEType	"Atom ID Codes" (IV-2649)
kGraphicsFlagsGray	struct ParameterDataBehavior (IV-2328)
kGraphicsImporterDontDoGammaCorrection	function GraphicsImportGetFlags (I-634)
kGroupAlignText	struct ParameterDataBehavior (IV-2328)
kGroupMatrix	struct ParameterDataBehavior (IV-2328)
kGroupNoName	struct ParameterDataBehavior (IV-2328)
kGroupSurroundBox	struct ParameterDataBehavior (IV-2328)
kH261CodecType	"Codec Identifiers" (IV-2655)
kH263CodecType	"Codec Identifiers" (IV-2655)
kHighLevelEvent	struct EventRecord (IV-2246)
kHighQuality	"Sound Device Features" (IV-2692)
kHSLColorBalanceImageFilterType	"Effects Codes" (IV-2662)
kICMPixelFormatIsIndexed	struct ICMPixelFormatInfo (IV-2282)
kICMPixelFormatIsPlanarMask	struct ICMPixelFormatInfo (IV-2282)
kICMPixelFormatIsSupportedByQD	struct ICMPixelFormatInfo (IV-2282)
kIMACompression	"Sound Formats" (IV-2692)
kImplodeTransitionType	"Effects Codes" (IV-2662)
kIndeo4CodecType	"Codec Identifiers" (IV-2655)
kInitializeQTMLDisableDirectSound	function InitializeQTML (II-756)
kInitializeQTMLNoSoundFlag	function InitializeQTML (II-756)
kInitializeQTMLUseExclusiveFullScreenModeFlag	

kInitializeQTMLUseGDIFlag	function InitializeQTML (II-756)
kInitialRotationAtom	function InitializeQTML (II-756)
kInputMapSubInputID	"Atom ID Codes" (IV-2649)
kInstKnobMissingDefault	"Atom ID Codes" (IV-2649)
kInstKnobMissingUnknown	struct InstKnobList (IV-2292)
kInstrumentExactMatch	struct InstKnobList (IV-2292)
kInstrumentMatchGMNumber	struct GMInstrumentInfo (IV-2272)
kInstrumentMatchName	function MusicFindTone (II-981)
kInstrumentMatchNumber	function MusicFindTone (II-981)
kInstrumentMatchSynthesizerName	function MusicFindTone (II-981)
kInstrumentMatchSynthesizerType	function MusicFindTone (II-981)
kInstrumentQualityField	struct GMInstrumentInfo (IV-2272)
kInstrumentRecommendedSubstitute	struct GMInstrumentInfo (IV-2272)
kInstrumentRoland8BitQuality	struct GMInstrumentInfo (IV-2272)
kIrisTransitionType	"Effects Codes" (IV-2662)
kITextAtomType	"Atom ID Codes" (IV-2649)
kITextStringAtomType	"Atom ID Codes" (IV-2649)
kJPEGCodecType	"Codec Identifiers" (IV-2655)
kKnobFixedPoint16	struct KnobDescription (IV-2299)
kKnobFixedPoint8	struct KnobDescription (IV-2299)
kKnobGroupStart	struct KnobDescription (IV-2299)
kKnobInterruptUnsafe	struct KnobDescription (IV-2299)
kKnobKeyrangeOverride	struct KnobDescription (IV-2299)
kKnobReadOnly	struct KnobDescription (IV-2299)
kKnobTypeBoolean	struct KnobDescription (IV-2299)
kKnobTypeButton	struct KnobDescription (IV-2299)
kKnobTypeGroupName	struct KnobDescription (IV-2299)
kKnobTypeHertz	struct KnobDescription (IV-2299)
kKnobTypeInstrument	struct KnobDescription (IV-2299)
kKnobTypeMilliseconds	struct KnobDescription (IV-2299)
kKnobTypeNote	struct KnobDescription (IV-2299)
kKnobTypeNumber	struct KnobDescription (IV-2299)
kKnobTypePan	struct KnobDescription (IV-2299)
kKnobTypePercentage	struct KnobDescription (IV-2299)
kKnobTypeSetting	struct KnobDescription (IV-2299)
kkQTVRDownLeft	function QTVRInteractionNudge (II-1389)
kLensFlareImageFilterType	"Effects Codes" (IV-2662)
kListElementDataType	"Atom ID Codes" (IV-2649)
kListElementType	"Atom ID Codes" (IV-2649)
kLittleEndianFormat	"Sound Formats" (IV-2692)
kM68kISA	struct RoutineRecord (IV-2405)
kMACE3Compression	"Sound Formats" (IV-2692)
kMACE6Compression	"Sound Formats" (IV-2692)
kMacPaintCodecType	"Sound Formats" (IV-2692)
kMarkerEventBeat	"Codec Identifiers" (IV-2655)
kMarkerEventEnd	atom 'wtxt' (IV-2616)
kMarkerEventTempo	atom 'wtxt' (IV-2616)
'kmat'	atom 'wtxt' (IV-2616)
	atom 'kmat' (IV-2554)

kMatrixTransitionType	"Effects Codes" (IV-2662)
kMediaVideoParamBlackLevel	function MediaGetVideoParam (II-868)
kMediaVideoParamBrightness	function MediaGetVideoParam (II-868)
kMediaVideoParamContrast	function MediaGetVideoParam (II-868)
kMediaVideoParamHue	function MediaGetVideoParam (II-868)
kMediaVideoParamSaturation	function MediaGetVideoParam (II-868)
kMediaVideoParamSharpness	function MediaGetVideoParam (II-868)
kMediaVideoParamWhiteLevel	function MediaGetVideoParam (II-868)
kMicrosoftADPCMFormat	"Sound Formats" (IV-2692)
kMotionJPEGBCodecType	"Codec Identifiers" (IV-2655)
kMIDIImport20Playable	function MIDIImportGetSettings (II-917)
kMIDIImportSilenceAfter	function MIDIImportGetSettings (II-917)
kMIDIImportSilenceBefore	function MIDIImportGetSettings (II-917)
kMIDIImportWantLyrics	function MIDIImportGetSettings (II-917)
kMotionJPEGACodecType	"Codec Identifiers" (IV-2655)
kMotionJPEGBCodecType	"Codec Identifiers" (IV-2655)
kMovieAnchorDataRefIsDefault	function GetMovieAnchorDataRef (I-458)
kMovieExportAbsoluteTime	function TextExportGetSettings (III-1928)
kMovieExportRelativeTime	function TextExportGetSettings (III-1928)
kMovieExportTextOnly	function TextExportGetSettings (III-1928)
kMovieLoadStateComplete	function GetMovieLoadState (I-477)
kMovieLoadStateError	function GetMovieLoadState (I-477)
kMovieLoadStateLoading	function GetMovieLoadState (I-477)
kMovieLoadStatePlayable	function GetMovieLoadState (I-477)
kMovieMediaAltText	"Atom ID Codes" (IV-2649)
kMovieMediaAutoPlay	"Atom ID Codes" (IV-2649)
kMovieMediaClipBegin	"Atom ID Codes" (IV-2649)
kMovieMediaClipDuration	"Atom ID Codes" (IV-2649)
kMovieMediaDataReference	"Atom ID Codes" (IV-2649)
kMovieMediaEnableFrameStepping	"Atom ID Codes" (IV-2649)
kMovieMediaHeight	"Atom ID Codes" (IV-2649)
kMovieMediaLeft	"Atom ID Codes" (IV-2649)
kMovieMediaLoop	"Atom ID Codes" (IV-2649)
kMovieMediaRectangleAtom	"Atom ID Codes" (IV-2649)
kMovieMediaRegionAtom	"Atom ID Codes" (IV-2649)
kMovieMediaSlaveAudio	"Atom ID Codes" (IV-2649)
kMovieMediaSlaveTrackDuration	"Atom ID Codes" (IV-2649)
kMovieMediaSpatialAdjustment	"Atom ID Codes" (IV-2649)
kMovieMediaTitle	"Atom ID Codes" (IV-2649)
kMovieMediaTop	"Atom ID Codes" (IV-2649)
kMovieMediaUseMIMETYPE	"Atom ID Codes" (IV-2649)
kMovieMediaWidth	"Atom ID Codes" (IV-2649)
kMoviePropertyDuration	"Atom ID Codes" (IV-2649)
kMoviePropertyNaturalBounds	"Atom ID Codes" (IV-2649)
kMoviePropertyTimeScale	"Atom ID Codes" (IV-2649)
kMPEGLayer3Format	"Sound Formats" (IV-2692)
kMpegYUV420CodecType	"Codec Identifiers" (IV-2655)
kMusicLoopTypeNormal	struct InstSampleDescRec (IV-2296)
kMusicLoopTypePalindrome	struct InstSampleDescRec (IV-2296)

kMusicPacketPortFound	struct MusicMIDIPacket (IV-2320)
kMusicPacketPortLost	struct MusicMIDIPacket (IV-2320)
kMusicPacketTimeGap	struct MusicMIDIPacket (IV-2320)
kNameAtom	“Atom ID Codes” (IV-2649)
kNonLinearTweenHeader	“Atom ID Codes” (IV-2649)
kNonRealTime	“Sound Device Features” (IV-2692)
kNoSoundComponentChain	function SoundComponentPlaySourceBuffer (III-1854)
kNoteRequestNoGM	struct NoteRequestInfo (IV-2323)
kNoteRequestNoSynthType	struct NoteRequestInfo (IV-2323)
kNoVolume	function GetTrackVolume (I-547)
kOffsetBinary	“Sound Formats” (IV-2692)
kOnlyDrawToSpriteWorld	function SpriteWorldIdle (III-1908)
kOpenDMLJPEGCodecType	“Codec Identifiers” (IV-2655)
kOperandAtomType	“Atom ID Codes” (IV-2649)
kOperatorAtomType	“Atom ID Codes” (IV-2649)
kParameterItemCheckBox	atom 'wtxt' (IV-2616)
kParameterItemColorPicker	atom 'wtxt' (IV-2616)
kParameterItemControl	atom 'wtxt' (IV-2616)
kParameterItemDragImage	atom 'wtxt' (IV-2616)
kParameterItemEditDouble	atom 'wtxt' (IV-2616)
kParameterItemEditFixed	atom 'wtxt' (IV-2616)
kParameterItemEditLong	atom 'wtxt' (IV-2616)
kParameterItemEditText	atom 'wtxt' (IV-2616)
kParameterItemGroupDivider	atom 'wtxt' (IV-2616)
kParameterItemLine	atom 'wtxt' (IV-2616)
kParameterItemPopUp	atom 'wtxt' (IV-2616)
kParameterItemRadioCluster	atom 'wtxt' (IV-2616)
kParameterItemStaticText	atom 'wtxt' (IV-2616)
kParameterValidationFinalValidation	
	function ImageCodecValidateParameters (II-745)
kParameterValidationNoFlags	
	function ImageCodecValidateParameters (II-745)
kPassThrough	function SoundComponentPlaySourceBuffer (III-1854)
kPhotoCDCCodecType	“Codec Identifiers” (IV-2655)
kPickDontMix	function NAPickEditInstrument (II-1033)
kPickEditAllowPick	function NAPickEditInstrument (II-1033)
kPickSameSynth	function NAPickEditInstrument (II-1033)
kPickUserInsts	function NAPickEditInstrument (II-1033)
kPlanarRGBCodecType	“Codec Identifiers” (IV-2655)
kPNGCodecType	“Codec Identifiers” (IV-2655)
kPopupStoreAsString	struct ParameterDataBehavior (IV-2328)
kPowerPCISA	struct RoutineRecord (IV-2405)
kProcDescriptorIsRelative	struct RoutineRecord (IV-2405)
kPushTransitionType	“Effects Codes” (IV-2662)
kQDesignCompression	“Sound Formats” (IV-2692)
kQTCOLORSyncProfile	“Atom ID Codes” (IV-2649)
kQTCPU Speed1Rating	struct QTA1tCPURatingRecord (IV-2346)
kQTCPU Speed2Rating	struct QTA1tCPURatingRecord (IV-2346)
kQTCPU Speed3Rating	struct QTA1tCPURatingRecord (IV-2346)

kQTCPU Speed4Rating	struct QTAItCPURatingRecord (IV-2346)
kQTCPU Speed5Rating	struct QTAItCPURatingRecord (IV-2346)
kQTDontRecompress	"Atom ID Codes" (IV-2649)
kQTEventRecordAtomType	"Atom ID Codes" (IV-2649)
kQTEventType	"Atom ID Codes" (IV-2649)
kQTInterlaceStyle	"Atom ID Codes" (IV-2649)
kQTMLHandlePortEvents	function CreatePortAssociation (I-148)
kQTMLNoIdleEvents	function CreatePortAssociation (I-148)
kQTParseTextHREFBaseURL	"Atom ID Codes" (IV-2649)
kQTParseTextHREFClickPoint	"Atom ID Codes" (IV-2649)
kQTParseTextHREFDelimiter	"Atom ID Codes" (IV-2649)
kQTParseTextHREFHREF	"Atom ID Codes" (IV-2649)
kQTParseTextHREFIsAutoHREF	"Atom ID Codes" (IV-2649)
kQTParseTextHREFRecomposeHREF	"Atom ID Codes" (IV-2649)
kQTParseTextHREFTarget	"Atom ID Codes" (IV-2649)
kQTParseTextHREFUseAltDelim	"Atom ID Codes" (IV-2649)
kQTPNGInterlaceAdam7	function GraphicsExportGetInterlaceStyle (I-575)
kQTPNGInterlaceNone	function GraphicsExportGetInterlaceStyle (I-575)
kQTResolutionSettings	"Atom ID Codes" (IV-2649)
kQTSAnnotationsChangedNotification	"Atom ID Codes" (IV-2649)
kQTSBandwidthAlertNotification	"Atom ID Codes" (IV-2649)
kQTSBufferTimeAID	"Atom ID Codes" (IV-2649)
kQTSClipRectAID	"Atom ID Codes" (IV-2649)
kQTSConnectionAtomType	"Atom ID Codes" (IV-2649)
kQTSConnectionMethodPrefsType	"Atom ID Codes" (IV-2649)
kQTSConnectionPrefsType	"Atom ID Codes" (IV-2649)
kQTSConnectionPrefsVersion	"Atom ID Codes" (IV-2649)
kQTSDirectConnectPrefsType	"Atom ID Codes" (IV-2649)
kQTS DontProxyDataType	"Atom ID Codes" (IV-2649)
kQTS DontProxyPrefsAtomType	"Atom ID Codes" (IV-2649)
kQTSDurationAID	"Atom ID Codes" (IV-2649)
kQTSDurationInfo	"Atom ID Codes" (IV-2649)
kQTSErrorNotification	"Atom ID Codes" (IV-2649)
kQTSHTTPProxyPrefsType	"Atom ID Codes" (IV-2649)
kQTSHTTPTransportType	"Atom ID Codes" (IV-2649)
kQTSLostPercentInfo	"Atom ID Codes" (IV-2649)
kQTSMediaDescriptionTextAID	"Atom ID Codes" (IV-2649)
kQTSMediaStreamAID	"Atom ID Codes" (IV-2649)
kQTSMediaStreamHeaderAID	"Atom ID Codes" (IV-2649)
kQTSMediaTypeInfo	"Atom ID Codes" (IV-2649)
kQTSNameInfo	"Atom ID Codes" (IV-2649)
kQTSNewPresentationNotification	"Atom ID Codes" (IV-2649)
kQTSNormalStatusDimensionsInfo	"Atom ID Codes" (IV-2649)
kQTSPrerollAckNotification	"Atom ID Codes" (IV-2649)
kQTS PresDoneChangingNotification	"Atom ID Codes" (IV-2649)
kQTSPresentationAID	"Atom ID Codes" (IV-2649)
kQTSPresentationChangedNotification	"Atom ID Codes" (IV-2649)
kQTSPresentationGoneNotification	"Atom ID Codes" (IV-2649)
kQTSPresentationHeaderAID	"Atom ID Codes" (IV-2649)

kQTSPProxyPrefsAtomType	"Atom ID Codes" (IV-2649)
kQTSRTSPProxyPrefsType	"Atom ID Codes" (IV-2649)
kQTSServerNameNotification	"Atom ID Codes" (IV-2649)
kQTSSOCKSPrefsType	"Atom ID Codes" (IV-2649)
kQTSSOCKSPProxyPrefsType	"Atom ID Codes" (IV-2649)
kQTSSourceBoundingRectInfo	function RTPMPSetInfo (III-1475)
kQTSSourceClipRectInfo	function RTPMPSetInfo (III-1475)
kQTSSourceDimensionsInfo	function RTPMPSetInfo (III-1475)
kQTSSourceGraphicsModeInfo	function RTPMPSetInfo (III-1475)
kQTSSourceInputMapInfo	function RTPMPSetInfo (III-1475)
kQTSSourceLanguageInfo	function RTPMPSetInfo (III-1475)
kQTSSourceLayerInfo	function RTPMPSetInfo (III-1475)
kQTSSourceMatrixInfo	function RTPMPSetInfo (III-1475)
kQTSSourceScaleInfo	function RTPMPSetInfo (III-1475)
kQTSSourceTrackFlagsInfo	function RTPMPSetInfo (III-1475)
kQTSSourceTrackIDInfo	function RTPMPSetInfo (III-1475)
kQTSSourceUserDataInfo	function RTPMPSetInfo (III-1475)
kQTSSourceVolumesInfo	function RTPMPSetInfo (III-1475)
kQTSStartAckNotification	"Atom ID Codes" (IV-2649)
kQTSStatHelperReturnPascalStringsFlag	
	struct QTSStatHelperNextParams (IV-2382)
kQTSStatisticsInfo	"Atom ID Codes" (IV-2649)
kQTSStatisticsNameAtomType	"Atom ID Codes" (IV-2649)
kQTSStatisticsStreamAtomType	"Atom ID Codes" (IV-2649)
kQTSStatisticsUnitsAtomType	"Atom ID Codes" (IV-2649)
kQTSStatisticsUnitsNameAtomType	"Atom ID Codes" (IV-2649)
kQTSStatusNotification	"Atom ID Codes" (IV-2649)
kQTSSStreamBeginChangingNotification	"Atom ID Codes" (IV-2649)
kQTSSStreamDoneChangingNotification	"Atom ID Codes" (IV-2649)
kQTSSStreamMediaType	"Atom ID Codes" (IV-2649)
kQTSTargetBufferDurationInfo	"Atom ID Codes" (IV-2649)
kQTSTCPTTransportType	"Atom ID Codes" (IV-2649)
kQTSTotalDataRateInfo	"Atom ID Codes" (IV-2649)
kQTSTotalDataRateInInfo	"Atom ID Codes" (IV-2649)
kQTSTotalDataRateOutInfo	"Atom ID Codes" (IV-2649)
kQTSTransAndProxyAtomType	"Atom ID Codes" (IV-2649)
kQTSTransportPrefsAtomType	"Atom ID Codes" (IV-2649)
kQTSUDPTTransportType	"Atom ID Codes" (IV-2649)
kQTTargetDataSize	"Atom ID Codes" (IV-2649)
kQTTIFFCompression_None	
	function GraphicsExportSetCompressionMethod (I-589)
kQTTIFFCompression_PackBits	
	function GraphicsExportSetCompressionMethod (I-589)
kQTVRA11HotSpots	function QTVREnableHotSpot (II-1340)
kQTVRA11Modes	function QTVRBeginUpdateStream (II-1335)
kQTVRAngleRangeAtomType	"Atom ID Codes" (IV-2649)
kQTVRBackBufferAlwaysRefresh	struct QTVRAreaOfInterest (IV-2392)
kQTVRBackBufferEveryIdle	struct QTVRAreaOfInterest (IV-2392)
kQTVRBackBufferEveryUpdate	struct QTVRAreaOfInterest (IV-2392)

kQTVRCantPanDown	function QTVRGetConstraintStatus (II-1353)
kQTVRCantPanLeft	function QTVRGetConstraintStatus (II-1353)
kQTVRCantPanRight	function QTVRGetConstraintStatus (II-1353)
kQTVRCantPanUp	function QTVRGetConstraintStatus (II-1353)
kQTVRCantTranslateDown	function QTVRGetConstraintStatus (II-1353)
kQTVRCantTranslateLeft	function QTVRGetConstraintStatus (II-1353)
kQTVRCantTranslateRight	function QTVRGetConstraintStatus (II-1353)
kQTVRCantTranslateUp	function QTVRGetConstraintStatus (II-1353)
kQTVRCantZoomIn	function QTVRGetConstraintStatus (II-1353)
kQTVRCantZoomOut	function QTVRGetConstraintStatus (II-1353)
kQTVRCanZoom	function QTVRGetControlSetting (II-1355)
kQTVRColorCursorAtomType	"Atom ID Codes" (IV-2649)
kQTVRColorCursorType	struct QTVRCursorRecord (IV-2395)
kQTVRCurrent	function QTVRGetViewState (II-1382)
kQTVRCurrentMode	function QTVRUpdate (II-1445)
kQTVRCurrentNode	function QTVRGoToNodeID (II-1386)
kQTVRCursorAtomType	"Atom ID Codes" (IV-2649)
kQTVRCursorParentAtomType	"Atom ID Codes" (IV-2649)
kQTVRDefault	function QTVRGetViewState (II-1382)
kQTVRDefaultNode	function QTVRGoToNodeID (II-1386)
kQTVRDefaultRes	function QTVRGetAvailableResolutions (II-1346)
kQTVRDegrees	function QTVRGetAngularUnits (II-1344)
kQTVRDontLoopViewFrames	function QTVRGetAnimationSetting (II-1344)
kQTVRDown	function QTVRInteractionNudge (II-1389)
kQTVRDownRight	function QTVRInteractionNudge (II-1389)
kQTVRFieldOfView	function QTVRGetConstraints (II-1352)
kQTVRFOVConstraintAtomType	"Atom ID Codes" (IV-2649)
kQTVRFullCache	function QTVRGetBackBufferMemInfo (II-1347)
kQTVRFullCorrection	function QTVRGetImagingProperty (II-1365)
kQTVRFullRes	function QTVRGetAvailableResolutions (II-1346)
kQTVRGetHotSpotTypeSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRHalfRes	function QTVRGetAvailableResolutions (II-1346)
kQTVRHotSpotAtomType	"Atom ID Codes" (IV-2649)
kQTVRHotSpotID	function QTVREnableHotSpot (II-1340)
kQTVRHotSpotInfoAtomType	"Atom ID Codes" (IV-2649)
kQTVRHotSpotParentAtomType	"Atom ID Codes" (IV-2649)
kQTVRHotSpotTrackRefAtomType	"Atom ID Codes" (IV-2649)
kQTVRHotSpotType	function QTVREnableHotSpot (II-1340)
kQTVRImageTrackRefAtomType	"Atom ID Codes" (IV-2649)
kQTVRImagingCorrection	function QTVRGetImagingProperty (II-1365)
kQTVRImagingCurrentMode	function QTVRGetImagingProperty (II-1365)
kQTVRImagingDefaultValue	function QTVRGetImagingProperty (II-1365)
kQTVRImagingDirectDraw	function QTVRGetImagingProperty (II-1365)
kQTVRImagingParentAtomType	"Atom ID Codes" (IV-2649)
kQTVRImagingQuality	function QTVRGetImagingProperty (II-1365)
kQTVRInteractionMouseClickHysteresis	function QTVRGetInteractionProperty (II-1368)
kQTVRInteractionMouseClickTimeout	function QTVRGetInteractionProperty (II-1368)

kQTVRInteractionMouseMoveScale	function QTVRGetInteractionProperty (II-1368)
kQTVRInteractionNudgeMode	function QTVRGetInteractionProperty (II-1368)
kQTVRInteractionPanTiltSpeed	function QTVRGetInteractionProperty (II-1368)
kQTVRInteractionTranslateOnMouseDown	function QTVRGetInteractionProperty (II-1368)
kQTVRInteractionZoomSpeed	function QTVRGetInteractionProperty (II-1368)
kQTVRLeft	function QTVRInteractionNudge (II-1389)
kQTVRLinkInfoAtomType	"Atom ID Codes" (IV-2649)
kQTVRMinimumCache	function QTVRGetBackBufferMemInfo (II-1347)
kQTVRMotion	function QTVRBeginUpdateStream (II-1335)
kQTVRMouseDown	function QTVRGetViewState (II-1382)
kQTVRMouseDownSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRMouseEnterSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRMouseLeaveSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRMouseStillDownSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRMouseUpSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRMouseWithinSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRNoCorrection	function QTVRGetImagingProperty (II-1365)
kQTVRNodeHeaderAtomType	"Atom ID Codes" (IV-2649)
kQTVRNodeIDAtomType	"Atom ID Codes" (IV-2649)
kQTVRNodeLocationAtomType	"Atom ID Codes" (IV-2649)
kQTVRNodeParentAtomType	"Atom ID Codes" (IV-2649)
kQTVRNudgeRotate	function QTVRGetInteractionProperty (II-1368)
kQTVRNudgeTranslate	function QTVRGetInteractionProperty (II-1368)
kQTVRObjectHotSpotTrackRefAtomID	"Atom ID Codes" (IV-2649)
kQTVRObjectImageTrackRefAtomID	"Atom ID Codes" (IV-2649)
kQTVRObjectImagingAtomType	"Atom ID Codes" (IV-2649)
kQTVRObjectInfoAtomID	"Atom ID Codes" (IV-2649)
kQTVRObjectInfoAtomType	"Atom ID Codes" (IV-2649)
kQTVRPalindromeViewFrames	function QTVRGetAnimationSetting (II-1344)
kQTVRPalindromeViews	function QTVRGetAnimationSetting (II-1344)
kQTVRPan	function QTVRGetConstraints (II-1352)
kQTVRPanConstraintAtomType	"Atom ID Codes" (IV-2649)
kQTVRPanning	function QTVRGetCurrentMouseMode (II-1358)
kQTVRPanoImagingAtomType	"Atom ID Codes" (IV-2649)
kQTVRPanoSampleDataAtomType	"Atom ID Codes" (IV-2649)
kQTVRPartialCorrection	function QTVRGetImagingProperty (II-1365)
kQTVRPlayEveryViewFrame	function QTVRGetAnimationSetting (II-1344)
kQTVRPlayStreamingViews	function QTVRGetAnimationSetting (II-1344)
kQTVRPreScreenEveryIdle	function QTVRSetPrescreenImagingCompleteProc (II-1432)
kQTVRPreviousNode	function QTVRGoToNodeID (II-1386)
kQTVRQuarterRes	function QTVRGetAvailableResolutions (II-1346)
kQTVRRadians	function QTVRGetAngularUnits (II-1344)
kQTVRReverseHControl	function QTVRGetControlSetting (II-1355)
kQTVRReverseVControl	function QTVRGetControlSetting (II-1355)
kQTVRRight	function QTVRInteractionNudge (II-1389)

kQTVRScrolling	function QTVRGetCurrentMouseMode (II-1358)
kQTVRSelecting	function QTVRGetCurrentMouseMode (II-1358)
kQTVRSetFieldOfViewSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRSetPanAngleSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRSetTiltAngleSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRSetViewCenterSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRStartFirstViewFrames	function QTVRGetAnimationSetting (II-1344)
kQTVRStatic	function QTVRBeginUpdateStream (II-1335)
kQTVRStdCursorType	struct QTVRCursorRecord (IV-2395)
kQTVRStringAtomType	"Atom ID Codes" (IV-2649)
kQTVRStringEncodingAtomType	"Atom ID Codes" (IV-2649)
kQTVRSuggestedCache	function QTVRGetBackBufferMemInfo (II-1347)
kQTVRSwapHVControl	function QTVRGetControlSetting (II-1355)
kQTVRSyncViewToFrameRate	function QTVRGetAnimationSetting (II-1344)
kQTVRTilt	function QTVRGetConstraints (II-1352)
kQTVRTiltConstraintAtomType	"Atom ID Codes" (IV-2649)
kQTVRTrackRefArrayAtomType	"Atom ID Codes" (IV-2649)
kQTVRTransitionDirection	function QTVRSetTransitionProperty (II-1435)
kQTVRTransitionSpeed	function QTVRSetTransitionProperty (II-1435)
kQTVRTransitionSwing	function QTVREnableTransition (II-1341)
kQTVRTranslating	function QTVRGetCurrentMouseMode (II-1358)
kQTVRTranslation	function QTVRGetControlSetting (II-1355)
kQTVRTriggerHotSpotSelector	function QTVRInstallInterceptProc (II-1387)
kQTVRUnconstrained	function QTVRGetConstraintStatus (II-1353)
kQTVRUNudgeSameAsMouse	function QTVRGetInteractionProperty (II-1368)
kQTVRUp	function QTVRInteractionNudge (II-1389)
kQTVRUpLeft	function QTVRInteractionNudge (II-1389)
kQTVRUpRight	function QTVRInteractionNudge (II-1389)
kQTVRUseDefaultCursor	struct QTVRCursorRecord (IV-2395)
kQTVRUseMovieGeometry	function QTVRGetBackBufferMemInfo (II-1347)
kQTVRVerticalCylinder	function QTVRGetBackBufferMemInfo (II-1347)
kQTVRViewCenterH	function QTVRWrapAndConstrain (II-1446)
kQTVRViewCenterV	function QTVRWrapAndConstrain (II-1446)
kQTVRWorldHeaderAtomType	"Atom ID Codes" (IV-2649)
kQTVRWrapPan	function QTVRGetControlSetting (II-1355)
kQTVRWrapTilt	function QTVRGetControlSetting (II-1355)
kQTVRZooming	function QTVRGetCurrentMouseMode (II-1358)
kQUALCOMMCompression	"Sound Formats" (IV-2692)
kQuickDrawCodecType	"Codec Identifiers" (IV-2655)
kQuickDrawGXCodecType	"Codec Identifiers" (IV-2655)
kRadialTransitionType	"Effects Codes" (IV-2662)
kRateConvert	"Sound Device Features" (IV-2692)
kRawCodecType	"Codec Identifiers" (IV-2655)
kReverse	"Sound Device Features" (IV-2692)
kRGBColorBalanceImageFilterType	"Effects Codes" (IV-2662)
kRoland8BitQuality	struct GMInstrumentInfo (IV-2272)
kRoutineIsDispatchedDefaultRoutine	struct RoutineRecord (IV-2405)
kRTPMPHasUserSettingsDialogCharacteristic	function RTPMPHasCharacteristic (III-1471)

kRTPMPMinPacketDuration	function RTPMPGetInfo (III-1464)
kRTPMPMinPayloadSize	function RTPMPGetInfo (III-1464)
kRTPMPNoSampleDataRequiredCharacteristic	function RTPMPHasCharacteristic (III-1471)
kRTPMPPayloadTypeDynamicFlag	struct RTPMPPayloadTypeParams (IV-2407)
kRTPMPPayloadTypeInfo	function RTPMPGetInfo (III-1464)
kRTPMPPayloadTypeStaticFlag	struct RTPMPPayloadTypeParams (IV-2407)
kRTPMPPrefersReliableTransportCharacteristic	function RTPMPHasCharacteristic (III-1471)
kRTPMPRealtimeModeFlag	function RTPMPInitialize (III-1473)
kRTPMPRequiredSampleDescriptionInfo	function RTPMPGetInfo (III-1464)
kRTPMPRequiresOutOfBandDimensionsCharacteristic	function RTPMPHasCharacteristic (III-1471)
kRTPMPRTTimeScaleInfo	function RTPMPGetInfo (III-1464)
kRTPMPStillProcessingData	function RTPMPIdle (III-1472)
kRTPMPSuggestedRepeatedPktSpacingInfo	function RTPMPGetInfo (III-1464)
kRTPMPSuggestedRepeatPktCountInfo	function RTPMPGetInfo (III-1464)
kRTPMPSyncSampleFlag	struct RTPMPSampleDataParams (IV-2407)
kRTPMPWantsMoreDataFlag	function RTPMPSetSampleData (III-1480)
kRTPRssmEveryPacketAChunkFlag	function RTPRssmGetCapabilities (III-1505)
kRTPRssmLostSomePackets	function RTPRssmSendPacketList (III-1518)
kRTPRssmNoReorderingRequiredFlag	function RTPRssmGetCapabilities (III-1505)
kRTPRssmQueueAndUseMarkerBitFlag	function RTPRssmGetCapabilities (III-1505)
kRTPRssmTrackLostPacketsFlag	function RTPRssmGetCapabilities (III-1505)
kScheduledSoundDoCallback	struct ExtendedScheduledSoundHeader (IV-2251)
kScheduledSoundDoScheduled	struct ExtendedScheduledSoundHeader (IV-2251)
kScheduledSoundExtendedHdr	struct ScheduledSoundHeader (IV-2419)
kScreenFloodMethodAlpha	struct CodecDecompressParams (IV-2190)
kScreenFloodMethodKeyColor	struct CodecDecompressParams (IV-2190)
kScreenFloodMethodNone	struct CodecDecompressParams (IV-2190)
kScriptIsJavaScript	struct QTDoScriptRecord (IV-2352)
kScriptIsLingoEvent	struct QTDoScriptRecord (IV-2352)
kScriptIsProjectorCommand	struct QTDoScriptRecord (IV-2352)
kScriptIsUnknownType	struct QTDoScriptRecord (IV-2352)
kScriptIsVBEvent	struct QTDoScriptRecord (IV-2352)
kSelectorsAreIndexable	struct RoutineDescriptor (IV-2404)
kSelectorsAreNotIndexable	struct RoutineDescriptor (IV-2404)
kSetAtomicInstCallerTosses	function NANewNoteChannelFromAtomicInstrument (II-1031)
kSetAtomicInstDontPreprocess	function NANewNoteChannelFromAtomicInstrument (II-1031)
kSetAtomicInstKeepOriginalInstrument	function NANewNoteChannelFromAtomicInstrument (II-1031)
kSetAtomicInstShareAcrossParts	function NANewNoteChannelFromAtomicInstrument (II-1031)
kSGICodecType	"Codec Identifiers" (IV-2655)
kSharpenImageFilterType	"Effects Codes" (IV-2662)

kSlideTransitionType	"Effects Codes" (IV-2662)
kSoftSynthComponentSubType	function InstrumentGetSynthesizerType (II-768)
kSolarizeImageFilterType	"Effects Codes" (IV-2662)
kSorensonCodecType	"Codec Identifiers" (IV-2655)
kSorensonYUV9CodecType	"Codec Identifiers" (IV-2655)
kSoundConverterDidntFillBuffer	function SoundConverterFillBuffer (III-1864)
kSoundConverterHasLeftOverData	function SoundConverterFillBuffer (III-1864)
kSoundNotCompressed	"Sound Formats" (IV-2692)
kSourcePaused	function SoundComponentPlaySourceBuffer (III-1854)
kSpriteAtomType	"Atom ID Codes" (IV-2649)
kSpriteBehaviorsAtomType	"Atom ID Codes" (IV-2649)
kSpriteCursorBehaviorAtomType	"Atom ID Codes" (IV-2649)
kSpriteFloatingPointVariableAtomType	"Atom ID Codes" (IV-2649)
kSpriteImageAtomType	"Atom ID Codes" (IV-2649)
kSpriteImageBehaviorAtomType	"Atom ID Codes" (IV-2649)
kSpriteImageDataAtomType	"Atom ID Codes" (IV-2649)
kSpriteImageDataRefAtomType	"Atom ID Codes" (IV-2649)
kSpriteImageDataRefTypeAtomType	"Atom ID Codes" (IV-2649)
kSpriteImageDefaultImageIndexAtomType	"Atom ID Codes" (IV-2649)
kSpriteImageGroupIDAtomType	"Atom ID Codes" (IV-2649)
kSpriteImageNameAtomType	"Atom ID Codes" (IV-2649)
kSpriteImageRegistrationAtomType	"Atom ID Codes" (IV-2649)
kSpriteImagesContainerAtomType	"Atom ID Codes" (IV-2649)
kSpriteNameAtomType	"Atom ID Codes" (IV-2649)
kSpritePropertyActionHandlingSpriteID	atom 'sprt' (IV-2584)
kSpritePropertyGraphicsMode	function GetSpriteProperty (I-512)
kSpritePropertyImageDataPtr	function GetSpriteProperty (I-512)
kSpritePropertyImageDescription	function GetSpriteProperty (I-512)
kSpritePropertyImageIndex	atom 'sprt' (IV-2584)
kSpritePropertyLayer	function GetSpriteProperty (I-512)
kSpritePropertyMatrix	function GetSpriteProperty (I-512)
kSpritePropertyVisible	function GetSpriteProperty (I-512)
kSpriteSharedDataAtomType	"Atom ID Codes" (IV-2649)
kSpriteStatusStringsBehaviorAtomType	"Atom ID Codes" (IV-2649)
kSpriteStringVariableAtomType	"Atom ID Codes" (IV-2649)
kSpriteUsesImageIDsAtomType	"Atom ID Codes" (IV-2649)
kSpriteVariablesContainerAtomType	"Atom ID Codes" (IV-2649)
kSpriteWorldDidDraw	function SpriteWorldIdle (III-1908)
kSpriteWorldNeedsToDraw	function SpriteWorldIdle (III-1908)
kSpriteWorldPreFlight	function SpriteWorldIdle (III-1908)
kStereoIn	"Sound Device Features" (IV-2692)
kStereoOut	"Sound Device Features" (IV-2692)
kSynthesizerConnectionFMS	struct SynthesizerConnections (IV-2464)
kSynthesizerConnectionMMgr	struct SynthesizerConnections (IV-2464)
kSynthesizerConnectionMono	struct SynthesizerConnections (IV-2464)
kSynthesizerConnectionOMS	struct SynthesizerConnections (IV-2464)
kSynthesizerConnectionQT	struct SynthesizerConnections (IV-2464)

kSynthesizerDynamicChannel	struct SynthesizerDescription (IV-2465)
kSynthesizerDynamicVoice	struct SynthesizerDescription (IV-2465)
kSynthesizerGM	struct SynthesizerDescription (IV-2465)
kSynthesizerHardware	struct SynthesizerDescription (IV-2465)
kSynthesizerHasSamples	struct SynthesizerDescription (IV-2465)
kSynthesizerHogsSystemChannel	struct SynthesizerDescription (IV-2465)
kSynthesizerMicrotone	struct SynthesizerDescription (IV-2465)
kSynthesizerMixedDrums	struct SynthesizerDescription (IV-2465)
kSynthesizerOffline	struct SynthesizerDescription (IV-2465)
kSynthesizerSlowSetPart	struct SynthesizerDescription (IV-2465)
kSynthesizerSoftware	struct SynthesizerDescription (IV-2465)
kSynthesizerUsesMIDIPort	struct SynthesizerDescription (IV-2465)
kTargaCodecType	“Codec Identifiers” (IV-2655)
kTargetChildMovieMovieID	“Atom ID Codes” (IV-2649)
kTargetChildMovieTrackID	“Atom ID Codes” (IV-2649)
kTargetChildMovieTrackName	“Atom ID Codes” (IV-2649)
kTargetMovieID	“Atom ID Codes” (IV-2649)
kTargetParentMovie	“Atom ID Codes” (IV-2649)
kTargetRootMovie	“Atom ID Codes” (IV-2649)
kTargetSpriteIndex	“Atom ID Codes” (IV-2649)
kTargetTrackID	“Atom ID Codes” (IV-2649)
kTargetTrackName	“Atom ID Codes” (IV-2649)
kTIFFCodecType	“Codec Identifiers” (IV-2655)
kTrackModifierInput	“Atom ID Codes” (IV-2649)
kTrackModifierInputName	“Atom ID Codes” (IV-2649)
kTrackModifierObjectID	“Atom ID Codes” (IV-2649)
kTrackModifierObjectQTEventSend	“Atom ID Codes” (IV-2649)
kTrackModifierPanAngle	“Atom ID Codes” (IV-2649)
kTrackModifierReference	“Atom ID Codes” (IV-2649)
kTrackModifierTiltAngle	“Atom ID Codes” (IV-2649)
kTrackModifierType	“Atom ID Codes” (IV-2649)
kTrackModifierVerticalFieldOfViewAngle	“Atom ID Codes” (IV-2649)
kTrackPropertyInstantiation	“Atom ID Codes” (IV-2649)
kTrackPropertyMediaType	“Atom ID Codes” (IV-2649)
kTrackReferenceChapterList	“Atom ID Codes” (IV-2649)
kTrackReferenceModifier	“Atom ID Codes” (IV-2649)
kTrackReferenceTimeCode	“Atom ID Codes” (IV-2649)
kTuneDontClipNotes	function TuneQueue (III-1964)
kTuneExcludeEdgeNotes	function TuneQueue (III-1964)
kTuneLoopUntil	function TuneQueue (III-1964)
kTuneMixMute	function TuneGetPartMix (III-1960)
kTuneMixSolo	function TuneGetPartMix (III-1960)
kTuneQuickStart	function TuneQueue (III-1964)
kTuneStartNewMaster	function TuneQueue (III-1964)
kTuneStartNow	function TuneQueue (III-1964)
kTween3dInitialCondition	“Atom ID Codes” (IV-2649)
kTweenData	“Atom ID Codes” (IV-2649)
kTweenDuration	“Atom ID Codes” (IV-2649)
kTweenEntry	“Atom ID Codes” (IV-2649)

kTweenFlags	"Atom ID Codes" (IV-2649)
kTweenInterpolationID	"Atom ID Codes" (IV-2649)
kTweenOutputMax	"Atom ID Codes" (IV-2649)
kTweenOutputMin	"Atom ID Codes" (IV-2649)
kTweenPictureData	"Atom ID Codes" (IV-2649)
kTweenRegionData	"Atom ID Codes" (IV-2649)
kTweenSequenceElement	"Atom ID Codes" (IV-2649)
kTweenStartOffset	"Atom ID Codes" (IV-2649)
kTweenType	"Atom ID Codes" (IV-2649)
kTweenType3dAngleAspectCameraData	"Atom ID Codes" (IV-2649)
kTweenType3dCameraData	"Atom ID Codes" (IV-2649)
kTweenType3dMatrix	"Atom ID Codes" (IV-2649)
kTweenType3dMatrixNonLinear	"Atom ID Codes" (IV-2649)
kTweenType3dQuaternion	"Atom ID Codes" (IV-2649)
kTweenType3dRotate	"Atom ID Codes" (IV-2649)
kTweenType3dRotateAboutAxis	"Atom ID Codes" (IV-2649)
kTweenType3dRotateAboutPoint	"Atom ID Codes" (IV-2649)
kTweenType3dRotateAboutVector	"Atom ID Codes" (IV-2649)
kTweenType3dScale	"Atom ID Codes" (IV-2649)
kTweenType3dSoundLocalizationData	"Atom ID Codes" (IV-2649)
kTweenType3dTranslate	"Atom ID Codes" (IV-2649)
kTweenType3dVRObject	"Atom ID Codes" (IV-2649)
kTweenTypeAtomList	"Atom ID Codes" (IV-2649)
kTweenTypeMultiMatrix	"Atom ID Codes" (IV-2649)
kTweenTypePathToFixedPoint	"Atom ID Codes" (IV-2649)
kTweenTypePathToMatrixRotation	"Atom ID Codes" (IV-2649)
kTweenTypePathToMatrixTranslation	"Atom ID Codes" (IV-2649)
kTweenTypePathToMatrixTranslationAndRotation	"Atom ID Codes" (IV-2649)
kTweenTypePathXtoY	"Atom ID Codes" (IV-2649)
kTweenTypePathYtoX	"Atom ID Codes" (IV-2649)
kTweenTypePolygon	"Atom ID Codes" (IV-2649)
kTweenTypeSpin	"Atom ID Codes" (IV-2649)
kTwosComplement	"Sound Formats" (IV-2692)
kULawCompression	"Sound Formats" (IV-2692)
kUseNativeISA	struct RoutineRecord (IV-2405)
kUseOptionalOutputDevice	function SndNewChannel (III-1839)
kUserDataMovieControllerType	"User Data Identifiers" (IV-2696)
kUserDataName	"User Data Identifiers" (IV-2696)
kUserDataTextAuthor	"User Data Identifiers" (IV-2696)
kUserDataTextChapter	"User Data Identifiers" (IV-2696)
kUserDataTextComment	"User Data Identifiers" (IV-2696)
kUserDataTextCopyright	"User Data Identifiers" (IV-2696)
kUserDataTextCreationDate	"User Data Identifiers" (IV-2696)
kUserDataTextDescription	"User Data Identifiers" (IV-2696)
kUserDataTextDirector	"User Data Identifiers" (IV-2696)
kUserDataTextDisclaimer	"User Data Identifiers" (IV-2696)
kUserDataTextEditDate1	"User Data Identifiers" (IV-2696)
kUserDataTextFullName	"User Data Identifiers" (IV-2696)
kUserDataTextHostComputer	"User Data Identifiers" (IV-2696)

kUserDataTextInformation	"User Data Identifiers" (IV-2696)
kUserDataTextKeywords	"User Data Identifiers" (IV-2696)
kUserDataTextMake	"User Data Identifiers" (IV-2696)
kUserDataTextModel	"User Data Identifiers" (IV-2696)
kUserDataTextOriginalFormat	"User Data Identifiers" (IV-2696)
kUserDataTextOriginalSource	"User Data Identifiers" (IV-2696)
kUserDataTextPerformers	"User Data Identifiers" (IV-2696)
kUserDataTextProducer	"User Data Identifiers" (IV-2696)
kUserDataTextProduct	"User Data Identifiers" (IV-2696)
kUserDataTextSoftware	"User Data Identifiers" (IV-2696)
kUserDataTextSpecialPlaybackRequirements	"User Data Identifiers" (IV-2696)
kUserDataTextWarning	"User Data Identifiers" (IV-2696)
kUserDataTextWriter	"User Data Identifiers" (IV-2696)
kUYVY422PixelFormat	"Pixel Formats" (IV-2686)
kVectorCodecType	"Codec Identifiers" (IV-2655)
kVersionCheckMask	struct QTAltVersionCheckRecord (IV-2348)
kVersionCheckMin	struct QTAltVersionCheckRecord (IV-2348)
kVideoCodecType	"Codec Identifiers" (IV-2655)
kVMAwareness	"Sound Device Features" (IV-2692)
kWaterRippleCodecType	"Codec Identifiers" (IV-2655)
kWhichAction	"Atom ID Codes" (IV-2649)
kWindowsRawCodecType	"Codec Identifiers" (IV-2655)
kWipeTransitionType	"Effects Codes" (IV-2662)
kYUV211PixelFormat	"Pixel Formats" (IV-2686)
kYUV411PixelFormat	"Pixel Formats" (IV-2686)
kYUV420CodecType	"Codec Identifiers" (IV-2655)
kYUVSPixelFormat	"Pixel Formats" (IV-2686)
kYUVUPixelFormat	"Pixel Formats" (IV-2686)
kYVU9PixelFormat	"Pixel Formats" (IV-2686)
kYVYU422PixelFormat	"Pixel Formats" (IV-2686)
kZoomTransitionType	"Effects Codes" (IV-2662)
langAfrikaans	"Localization Codes" (IV-2673)
langAlbanian	"Localization Codes" (IV-2673)
langAmharic	"Localization Codes" (IV-2673)
langArabic	"Localization Codes" (IV-2673)
langArmenian	"Localization Codes" (IV-2673)
langAssamese	"Localization Codes" (IV-2673)
langAymara	"Localization Codes" (IV-2673)
langAzerbaijanAr	"Localization Codes" (IV-2673)
langAzerbaijani	"Localization Codes" (IV-2673)
langBasque	"Localization Codes" (IV-2673)
langBelorussian	"Localization Codes" (IV-2673)
langBengali	"Localization Codes" (IV-2673)
langBreton	"Localization Codes" (IV-2673)
langBulgarian	"Localization Codes" (IV-2673)
langBurmese	"Localization Codes" (IV-2673)
langByelorussian	"Localization Codes" (IV-2673)
langCatalan	"Localization Codes" (IV-2673)
langChewa	"Localization Codes" (IV-2673)

langCroatian	"Localization Codes" (IV-2673)
langCzech	"Localization Codes" (IV-2673)
langDanish	"Localization Codes" (IV-2673)
langDutch	"Localization Codes" (IV-2673)
langDzongkha	"Localization Codes" (IV-2673)
langEnglish	"Localization Codes" (IV-2673)
langEsperanto	"Localization Codes" (IV-2673)
langEstonian	"Localization Codes" (IV-2673)
langFaroese	"Localization Codes" (IV-2673)
langFarsi	"Localization Codes" (IV-2673)
langFinnish	"Localization Codes" (IV-2673)
langFlemish	"Localization Codes" (IV-2673)
langFrench	"Localization Codes" (IV-2673)
langGalician	"Localization Codes" (IV-2673)
langGeorgian	"Localization Codes" (IV-2673)
langGerman	"Localization Codes" (IV-2673)
langGreek	"Localization Codes" (IV-2673)
langGreekPoly	"Localization Codes" (IV-2673)
langGreenlandic	"Localization Codes" (IV-2673)
langGuarani	"Localization Codes" (IV-2673)
langGujarati	"Localization Codes" (IV-2673)
langHebrew	"Localization Codes" (IV-2673)
langHindi	"Localization Codes" (IV-2673)
langHungarian	"Localization Codes" (IV-2673)
langIcelandic	"Localization Codes" (IV-2673)
langIndonesian	"Localization Codes" (IV-2673)
langInuktitut	"Localization Codes" (IV-2673)
langIrishGaelic	"Localization Codes" (IV-2673)
langIrishGaelic,	atom 'wtxt' (IV-2616)
langIrishGaelicScript	"Localization Codes" (IV-2673)
langIrishGaelicScript,	atom 'wtxt' (IV-2616)
langItalian	"Localization Codes" (IV-2673)
langJapanese	"Localization Codes" (IV-2673)
langJavaneseRom	"Localization Codes" (IV-2673)
langKannada	"Localization Codes" (IV-2673)
langKashmiri	"Localization Codes" (IV-2673)
langKazakh	"Localization Codes" (IV-2673)
langKhmer	"Localization Codes" (IV-2673)
langKinyarwanda	"Localization Codes" (IV-2673)
langKirghiz	"Localization Codes" (IV-2673)
langKorean	"Localization Codes" (IV-2673)
langKurdish	"Localization Codes" (IV-2673)
langLao	"Localization Codes" (IV-2673)
langLatin	"Localization Codes" (IV-2673)
langLatvian	"Localization Codes" (IV-2673)
langLithuanian	"Localization Codes" (IV-2673)
langMacedonian	"Localization Codes" (IV-2673)
langMalagasy	"Localization Codes" (IV-2673)
langMalayalam	"Localization Codes" (IV-2673)

langMalayArabic	"Localization Codes" (IV-2673)
langMalayRoman	"Localization Codes" (IV-2673)
langMaltese	"Localization Codes" (IV-2673)
langManxGaelic	"Localization Codes" (IV-2673)
langMarathi	"Localization Codes" (IV-2673)
langMoldavian	"Localization Codes" (IV-2673)
langMongolian	"Localization Codes" (IV-2673)
langMongolianCyr	"Localization Codes" (IV-2673)
langNepali	"Localization Codes" (IV-2673)
langNorwegian	"Localization Codes" (IV-2673)
langNyanja	"Localization Codes" (IV-2673)
langOriya	"Localization Codes" (IV-2673)
langOromo	"Localization Codes" (IV-2673)
langPashto	"Localization Codes" (IV-2673)
langPersian	"Localization Codes" (IV-2673)
langPolish	"Localization Codes" (IV-2673)
langPortuguese	"Localization Codes" (IV-2673)
langPunjabi	"Localization Codes" (IV-2673)
langQuechua	"Localization Codes" (IV-2673)
langRomanian	"Localization Codes" (IV-2673)
langRuanda	"Localization Codes" (IV-2673)
langRundi	"Localization Codes" (IV-2673)
langRussian	"Localization Codes" (IV-2673)
langSami	"Localization Codes" (IV-2673)
langSanskrit	"Localization Codes" (IV-2673)
langScottishGaelic	"Localization Codes" (IV-2673)
langSerbian	"Localization Codes" (IV-2673)
langSimpChinese	"Localization Codes" (IV-2673)
langSimpChinese,	atom 'wtxt' (IV-2616)
langSindhi	"Localization Codes" (IV-2673)
langSinhalese	"Localization Codes" (IV-2673)
langSlovak	"Localization Codes" (IV-2673)
langSlovenian	"Localization Codes" (IV-2673)
langSomali	"Localization Codes" (IV-2673)
langSpanish	"Localization Codes" (IV-2673)
langSundaneseRom	"Localization Codes" (IV-2673)
langSwahili	"Localization Codes" (IV-2673)
langSwedish	"Localization Codes" (IV-2673)
langTagalog	"Localization Codes" (IV-2673)
langTajiki	"Localization Codes" (IV-2673)
langTamil	"Localization Codes" (IV-2673)
langTatar	"Localization Codes" (IV-2673)
langTelugu	"Localization Codes" (IV-2673)
langThai	"Localization Codes" (IV-2673)
langTibetan	"Localization Codes" (IV-2673)
langTigrinya	"Localization Codes" (IV-2673)
langTongan	"Localization Codes" (IV-2673)
langTradChinese	"Localization Codes" (IV-2673)
langTradChinese,	atom 'wtxt' (IV-2616)

langTurkish	"Localization Codes" (IV-2673)
langTurkmen	"Localization Codes" (IV-2673)
langUighur	"Localization Codes" (IV-2673)
langUkrainian	"Localization Codes" (IV-2673)
langUnspecified	"Localization Codes" (IV-2673)
langUrdu	"Localization Codes" (IV-2673)
langUzbek	"Localization Codes" (IV-2673)
langVietnamese	"Localization Codes" (IV-2673)
langWelsh	"Localization Codes" (IV-2673)
langYiddish	"Localization Codes" (IV-2673)
leftOverBlockSize	struct LeftOverBlock (IV-2301)
limitReachedErr	"Error Codes" (IV-2663)
linearMatrixType	function GetMatrixType (I-419)
linearTranslateMatrixType	function GetMatrixType (I-419)
'link'	atom 'link' (IV-2556)
linkSoundComponentsCmd	"Sound Commands" (IV-2691)
'load'	atom 'load' (IV-2556)
loadBackwardTrackEdits	function LoadMediaIntoRam (II-779)
loadForwardTrackEdits	function LoadMediaIntoRam (II-779)
LoadSettingsAID	"Atom ID Codes" (IV-2649)
lockPortBitsBadPortErr	"Error Codes" (IV-2663)
lockPortBitsBadSurfaceErr	"Error Codes" (IV-2663)
lockPortBitsSurfaceLostErr	"Error Codes" (IV-2663)
lockPortBitsWindowClippedErr	"Error Codes" (IV-2663)
lockPortBitsWindowMovedErr	"Error Codes" (IV-2663)
lockPortBitsWindowResizedErr	"Error Codes" (IV-2663)
lockPortBitsWrongGDeviceErr	"Error Codes" (IV-2663)
'LOOP'	atom 'LOOP' (IV-2557)
loopTimeBase	function GetTimeBaseFlags (I-515)
'MAC3'	function GetCompressionInfo (I-398)
'MAC6'	function GetCompressionInfo (I-398)
magentaColor	"Color Constants" (IV-2657)
mainScreen	struct GDevice (IV-2264)
mapPix	function NewImageGWorld (II-1066)
mAtEnd	function MediaIdle (II-874)
matrixErr	atom 'wtxt' (IV-2616)
matrixFlagScale1x	struct CodecDecompressParams (IV-2190)
matrixFlagScale2x	struct CodecDecompressParams (IV-2190)
matrixFlagScaleHalf	struct CodecDecompressParams (IV-2190)
'matt'	atom 'matt' (IV-2557)
MatteAID	"Atom ID Codes" (IV-2649)
MatteCompAID	"Atom ID Codes" (IV-2649)
'maxr'	atom 'maxr' (IV-2558)
maxSizeToGrowTooSmall	"Error Codes" (IV-2663)
mcActionActivate	"Movie Controller Actions" (IV-2680)
mcActionAdjustCursor	"Movie Controller Actions" (IV-2680)
mcActionBadgeClick	"Movie Controller Actions" (IV-2680)
mcActionClickAndHoldPoint	"Movie Controller Actions" (IV-2680)
mcActionControllerSizeChanged	"Movie Controller Actions" (IV-2680)

mcActionCustomButtonClick	"Movie Controller Actions" (IV-2680)
mcActionDeactivate	"Movie Controller Actions" (IV-2680)
mcActionDoScript	"Movie Controller Actions" (IV-2680)
mcActionDraw	"Movie Controller Actions" (IV-2680)
mcActionEvaluateExpression	"Movie Controller Actions" (IV-2680)
mcActionExecuteAllActionsForQTEvent	"Movie Controller Actions" (IV-2680)
mcActionExecuteOneActionForQTEvent	"Movie Controller Actions" (IV-2680)
mcActionFetchParameterAs	"Movie Controller Actions" (IV-2680)
mcActionForceTimeTableUpdate	"Movie Controller Actions" (IV-2680)
mcActionGetChapterTime	"Movie Controller Actions" (IV-2680)
mcActionGetCursorByID	"Movie Controller Actions" (IV-2680)
mcActionGetCursorSettingEnabled	"Movie Controller Actions" (IV-2680)
mcActionGetDragEnabled	"Movie Controller Actions" (IV-2680)
mcActionGetExternalMovie	"Movie Controller Actions" (IV-2680)
mcActionGetFlags	"Movie Controller Actions" (IV-2680)
mcActionGetIndChapter	"Movie Controller Actions" (IV-2680)
mcActionGetKeysEnabled	"Movie Controller Actions" (IV-2680)
mcActionGetLooping	"Movie Controller Actions" (IV-2680)
mcActionGetLoopIsPalindrome	"Movie Controller Actions" (IV-2680)
mcActionGetNextURL	"Movie Controller Actions" (IV-2680)
mcActionGetPlayEveryFrame	"Movie Controller Actions" (IV-2680)
mcActionGetPlayRate	"Movie Controller Actions" (IV-2680)
mcActionGetPlaySelection	"Movie Controller Actions" (IV-2680)
mcActionGetSelectionBegin	"Movie Controller Actions" (IV-2680)
mcActionGetSelectionDuration	"Movie Controller Actions" (IV-2680)
mcActionGetTimeSliderRect	"Movie Controller Actions" (IV-2680)
mcActionGetUseBadge	"Movie Controller Actions" (IV-2680)
mcActionGetVolume	"Movie Controller Actions" (IV-2680)
mcActionGoToTime	"Movie Controller Actions" (IV-2680)
mcActionIdle	"Movie Controller Actions" (IV-2680)
mcActionKey	"Movie Controller Actions" (IV-2680)
mcActionLinkToURL	"Movie Controller Actions" (IV-2680)
mcActionMouseDown	"Movie Controller Actions" (IV-2680)
mcActionMovieChanged	"Movie Controller Actions" (IV-2680)
mcActionMovieClick	"Movie Controller Actions" (IV-2680)
mcActionMovieEdited	"Movie Controller Actions" (IV-2680)
mcActionPerformActionList	"Movie Controller Actions" (IV-2680)
mcActionPlay	"Movie Controller Actions" (IV-2680)
mcActionPrerollAndPlay	"Movie Controller Actions" (IV-2680)
mcActionRestartAtTime	"Movie Controller Actions" (IV-2680)
mcActionResume	"Movie Controller Actions" (IV-2680)
mcActionSetColorTable	"Movie Controller Actions" (IV-2680)
mcActionSetControllerKeysEnabled	"Movie Controller Actions" (IV-2680)
mcActionSetControllerTimeLimits	"Movie Controller Actions" (IV-2680)
mcActionSetCursorSettingEnabled	"Movie Controller Actions" (IV-2680)
mcActionSetDragEnabled	"Movie Controller Actions" (IV-2680)
mcActionSetFlags	"Movie Controller Actions" (IV-2680)
mcActionSetGrowBoxBounds	"Movie Controller Actions" (IV-2680)
mcActionSetKeysEnabled	"Movie Controller Actions" (IV-2680)

mcActionSetLooping	"Movie Controller Actions" (IV-2680)
mcActionSetLoopIsPalindrome	"Movie Controller Actions" (IV-2680)
mcActionSetPlayEveryFrame	"Movie Controller Actions" (IV-2680)
mcActionSetPlaySelection	"Movie Controller Actions" (IV-2680)
mcActionSetSelectionBegin	"Movie Controller Actions" (IV-2680)
mcActionSetSelectionDuration	"Movie Controller Actions" (IV-2680)
mcActionSetUseBadge	"Movie Controller Actions" (IV-2680)
mcActionSetVolume	"Movie Controller Actions" (IV-2680)
mcActionShowBalloon	"Movie Controller Actions" (IV-2680)
mcActionShowMessageString	"Movie Controller Actions" (IV-2680)
mcActionShowStatusString	"Movie Controller Actions" (IV-2680)
mcActionStep	"Movie Controller Actions" (IV-2680)
mcActionSuspend	"Movie Controller Actions" (IV-2680)
mcActionUseTrackForTimeTable	"Movie Controller Actions" (IV-2680)
mcFlagSuppressMovieFrame	atom 'wtxt' (IV-2616)
mcFlagSuppressSpeakerButton	atom 'wtxt' (IV-2616)
mcFlagSuppressStepButtons	atom 'wtxt' (IV-2616)
mcFlagsUseWindowPalette	atom 'wtxt' (IV-2616)
mcInfoClearAvailable	function MCGetControllerInfo (II-808)
mcInfoCopyAvailable	function MCGetControllerInfo (II-808)
mcInfoCutAvailable	function MCGetControllerInfo (II-808)
mcInfoEditingEnabled	function MCGetControllerInfo (II-808)
mcInfoHasSound	function MCGetControllerInfo (II-808)
mcInfoIsInPalindrome	function MCGetControllerInfo (II-808)
mcInfoIsLooping	function MCGetControllerInfo (II-808)
mcInfoIsPlaying	function MCGetControllerInfo (II-808)
mcInfoPasteAvailable	function MCGetControllerInfo (II-808)
mcInfoUndoAvailable	function MCGetControllerInfo (II-808)
mcMenuClear	function MCGetMenuString (II-814)
mcMenuCopy	function MCGetMenuString (II-814)
mcMenuCut	function MCGetMenuString (II-814)
mcMenuPaste	function MCGetMenuString (II-814)
mcMenuUndo	function MCGetMenuString (II-814)
mcNotVisible	function NewMovieController (II-1071)
mcPositionDontInvalidate	function MCPositionController (II-824)
mcScaleMovieToFit	function MCPositionController (II-824)
mcTopLeftMovie	function MCPositionController (II-824)
mcWithBadge	function NewMovieController (II-1071)
mcWithFrame	function NewMovieController (II-1071)
'mdat'	atom 'mdat' (IV-2558)
'mdhd'	atom 'mdhd' (IV-2559)
'mdia'	atom 'mdia' (IV-2560)
mDidDraw	function MediaIdle (II-874)
MediaAID	"Atom ID Codes" (IV-2649)
MediaHandlerType	"Component Identifiers" (IV-2657)
MediaHeaderAID	"Atom ID Codes" (IV-2649)
MediaInfoAID	"Atom ID Codes" (IV-2649)
mediaQualityBest	function GetMediaQuality (I-441)
mediaQualityBetter	function GetMediaQuality (I-441)

mediaQualityDraft	function GetMediaQuality (I-441)
mediaQualityNormal	function GetMediaQuality (I-441)
mediaSampleNotSync	function AddMediaSample (I-27)
mediaTypesDontMatch	“Error Codes” (IV-2663)
'micr'	atom 'wtxt' (IV-2616)
midiManagerAbsentOSErr	“Error Codes” (IV-2663)
'mill'	atom 'wtxt' (IV-2616)
milliseconds	function SPBRecord (III-1878)
'mime'	atom 'mime' (IV-2561)
'minf'(base)	atom 'minf'(base) (IV-2561)
'minf'(generic)	atom 'minf'(generic) (IV-2562)
'minf'(sound)	atom 'minf'(sound) (IV-2564)
'minf'(video)	atom 'minf'(video) (IV-2565)
missingRequiredParameterErr	“Error Codes” (IV-2663)
mMustDraw	function MediaIdle (II-874)
mNeedsToDraw	function MediaIdle (II-874)
'moov'	atom 'moov' (IV-2566)
mouseDown	struct EventRecord (IV-2246)
mouseMovedMessage	struct EventRecord (IV-2246)
mouseUp	struct EventRecord (IV-2246)
MovieAID	“Atom ID Codes” (IV-2649)
MovieBufferHintsAID	“Atom ID Codes” (IV-2649)
MovieControllerComponentType	“Component Identifiers” (IV-2657)
MovieDataAtomType	“Atom ID Codes” (IV-2649)
MovieDataRefAliasAID	“Atom ID Codes” (IV-2649)
movieDrawingCallAlways	function SetMovieDrawingCompleteProc (III-1617)
movieDrawingCallWhenChanged	function SetMovieDrawingCompleteProc (III-1617)
movieExportDuration	callback MovieExportGetPropertyProc (III-2102)
movieExportHeight	callback MovieExportGetPropertyProc (III-2102)
movieExportMustGetSourceMediaType	struct ComponentDescription (IV-2212)
movieExportNeedsResourceFork	struct ComponentDescription (IV-2212)
MovieExportType	“Component Identifiers” (IV-2657)
movieExportVideoFilter	callback MovieExportGetPropertyProc (III-2102)
movieExportWidth	callback MovieExportGetPropertyProc (III-2102)
movieFileSpecValid	function ConvertFileToMovieFile (I-129)
MovieHeaderAID	“Atom ID Codes” (IV-2649)
movieImportCreateTrack	function MovieImportFile (II-942)
movieImportInParallel	function MovieImportFile (II-942)
movieImportMustUseTrack	function MovieImportFile (II-942)
movieImportResultUsedMultipleTracks	function MovieImportFile (II-942)
movieImportSubTypeIsFileExtension	struct ComponentDescription (IV-2212)
MovieImportType	“Component Identifiers” (IV-2657)
movieInDataForkResID	function AddMovieResource (I-36)
MovieMediaType	“Component Identifiers” (IV-2657)
movieProgressClose	callback MovieProgressProc (III-2105)
movieProgressOpen	callback MovieProgressProc (III-2105)
movieProgressUpdatePercent	callback MovieProgressProc (III-2105)
MovieResourceAtomType	“Atom ID Codes” (IV-2649)

movieScrapDontZeroScrap	function PutMovieOnScrap (II-1153)
movieScrapOnlyPutMovie	function PutMovieOnScrap (II-1153)
movieTextNotFoundErr	"Error Codes" (IV-2663)
movieToFileOnlyExport	function ConvertFileToMovieFile (I-129)
movieToolboxUninitialized	"Error Codes" (IV-2663)
movieTrackCharacteristic	function GetMovieIndTrackType (I-475)
movieTrackEnabledOnly	function GetMovieIndTrackType (I-475)
movieTrackMediaType	function GetMovieIndTrackType (I-475)
mPartialDraw	function MediaIdle (II-874)
MPEGMediaType	"Component Identifiers" (IV-2657)
mPreflightDraw	function MediaIdle (II-874)
MusicMediaType	"Component Identifiers" (IV-2657)
'mvhd'	atom 'mvhd' (IV-2567)
'name'(sprite)	atom 'name'(sprite) (IV-2568)
'name'(userdata)	atom 'name'(userdata) (IV-2569)
'ndhd'	atom 'ndhd' (IV-2569)
newDepth	function NewImageGWorld (II-1066)
newMovieActive	function CreateMovieFile (I-145)
newMovieDontAskUnresolvedDataRefs	function NewMovieFromDataFork (II-1075)
newMovieDontAutoAlternate	function CreateMovieFile (I-145)
newMovieDontResolveDataRefs	function NewMovieFromDataFork (II-1075)
newRowBytes	function NewImageGWorld (II-1066)
nextTimeEdgeOK	function GetMediaNextInterestingTime (I-437)
nextTimeIgnoreActiveSegment	function GetMovieNextInterestingTime (I-480)
nextTimeMediaEdit	function GetMediaNextInterestingTime (I-437)
nextTimeMediaSample	function GetMediaNextInterestingTime (I-437)
nextTimeStep	function MediaGetNextStepTime (II-856)
nextTimeSyncSample	function GetMediaNextInterestingTime (I-437)
nextTimeTrackEdit	function GetMovieNextInterestingTime (I-480)
'nloc'	atom 'nloc' (IV-2570)
noCodecErr	"Error Codes" (IV-2663)
noDataHandler	"Error Codes" (IV-2663)
noDefaultDataRef	"Error Codes" (IV-2663)
noDefaultOpcodes	function StdPix (III-1912)
noDMAErr	atom 'wtxt' (IV-2616)
noDriver	struct GDevice (IV-2264)
noErr	function QTIsStandardParameterDialogEvent (II-1186)
noExportProcAvailableErr	"Error Codes" (IV-2663)
noMediaHandler	"Error Codes" (IV-2663)
noMemoryNodeFailedInitialize	"Error Codes" (IV-2663)
noMoreKeyColorsErr	atom 'wtxt' (IV-2616)
noMovieFound	"Error Codes" (IV-2663)
'NONE'	function GetCompressionInfo (I-398)
noNewDevice	function NewImageGWorld (II-1066)
nonMatchingEditState	"Error Codes" (IV-2663)
noPathMappingErr	"Error Codes" (IV-2663)
noRecordOfApp	atom 'wtxt' (IV-2616)
noSoundTrackInMovieErr	"Error Codes" (IV-2663)
noSourceTreeFoundErr	"Error Codes" (IV-2663)

notAllowedToSaveMovieErr	"Error Codes" (IV-2663)
notAQTVRMovieErr	"Error Codes" (IV-2663)
notCompressed	function GetCompressionInfo (I-398)
noteChannelNotAllocatedOSErr	"Error Codes" (IV-2663)
notEnoughDataErr	"Error Codes" (IV-2663)
notExactMatrixErr	atom 'wtxt' (IV-2616)
notExactSizeErr	atom 'wtxt' (IV-2616)
notImplementedMusicOSErr	"Error Codes" (IV-2663)
notLeafAtomErr	"Error Codes" (IV-2663)
notPatBic	"Graphics Transfer Modes" (IV-2670)
notPatCopy	"Graphics Transfer Modes" (IV-2670)
notPatOr	"Graphics Transfer Modes" (IV-2670)
notPatXor	"Graphics Transfer Modes" (IV-2670)
notSrcBic	"Graphics Transfer Modes" (IV-2670)
notSrcBic,	atom 'wtxt' (IV-2616)
notSrcCopy	"Graphics Transfer Modes" (IV-2670)
notSrcCopy,	atom 'wtxt' (IV-2616)
notSrcOr	"Graphics Transfer Modes" (IV-2670)
notSrcOr,	atom 'wtxt' (IV-2616)
notSrcXor	"Graphics Transfer Modes" (IV-2670)
notSrcXor,	atom 'wtxt' (IV-2616)
noVideoTrackInMovieErr	"Error Codes" (IV-2663)
ntscIn	function VDGetVBlankRect (III-2021)
nullCmd	"Sound Commands" (IV-2691)
nullEvent	struct EventRecord (IV-2246)
'nump'	atom 'nump' (IV-2570)
'obid'	atom 'obid' (IV-2571)
oddField1ToEvenFieldOut	
	function ImageCodecExtractAndCombineFields (II-705)
oddField1ToOddFieldOut	
	function ImageCodecExtractAndCombineFields (II-705)
oddField2ToEvenFieldOut	
	function ImageCodecExtractAndCombineFields (II-705)
oddField2ToOddFieldOut	
	function ImageCodecExtractAndCombineFields (II-705)
optionKey	function QTVRMouseDown (II-1391)
osEvt	struct EventRecord (IV-2246)
osEvtMessageMask	struct EventRecord (IV-2246)
palIn	function VDGetVBlankRect (III-2021)
palindromeLoopTimeBase	function GetTimeBaseFlags (I-515)
pasteInParallel	function PasteHandleIntoMovie (II-1137)
patBic	"Graphics Transfer Modes" (IV-2670)
patCopy	"Graphics Transfer Modes" (IV-2670)
pathNotVerifiedErr	"Error Codes" (IV-2663)
pathTooLongErr	"Error Codes" (IV-2663)
patOr	"Graphics Transfer Modes" (IV-2670)
patXor	"Graphics Transfer Modes" (IV-2670)
pauseCmd	"Sound Commands" (IV-2691)
'payt'	atom 'payt' (IV-2571)

```

pdActionActivateSubPanel
    function ImageCodecStandardParameterDialogDoAction (II-740)
pdActionConductStopAlert
    function QTStandardParameterDialogDoAction (II-1313)
pdActionConductStopAlert
    function ImageCodecStandardParameterDialogDoAction (II-740)
pdActionConfirmDialog
    function ImageCodecStandardParameterDialogDoAction (II-740)
pdActionGetDialogValues
    function ImageCodecStandardParameterDialogDoAction (II-740)
pdActionGetSubPanelMenu
    function ImageCodecStandardParameterDialogDoAction (II-740)
pdActionSetAppleMenu
    function ImageCodecStandardParameterDialogDoAction (II-740)
pdActionSetColorPickerEventProc
    function ImageCodecStandardParameterDialogDoAction (II-740)
pdActionSetDialogTitle
    function ImageCodecStandardParameterDialogDoAction (II-740)
pdActionSetEditMenu
    function ImageCodecStandardParameterDialogDoAction (II-740)
pdActionSetPreviewPicture
    function ImageCodecStandardParameterDialogDoAction (II-740)
pdActionSetPreviewUserItem
    function ImageCodecStandardParameterDialogDoAction (II-740)
'pdat'
    atom 'pdat' (IV-2572)
pdOptionsAllowOptionalInterpolations
    function ImageCodecCreateStandardParameterDialog (II-692)
pdOptionsCollectOneValue
    function ImageCodecCreateStandardParameterDialog (II-692)
perspectiveMatrixType
    function GetMatrixType (I-419)
pixelsLocked
    function NewImageGWorld (II-1066)
pixelsPurgeable
    function NewImageGWorld (II-1066)
pixPurge
    function NewImageGWorld (II-1066)
'pmax'
    atom 'pmax' (IV-2572)
'pnot'
    atom 'pnot' (IV-2573)
preloadAlways
    function GetTrackLoadSettings (I-532)
preloadOnlyIfEnabled
    function GetTrackLoadSettings (I-532)
progressOpAddMovieSelection
    callback MovieProgressProc (III-2105)
progressOpCopy
    callback MovieProgressProc (III-2105)
progressOpCut
    callback MovieProgressProc (III-2105)
progressOpExportMovie
    callback MovieProgressProc (III-2105)
progressOpFlatten
    callback MovieProgressProc (III-2105)
progressOpImportMovie
    callback MovieProgressProc (III-2105)
progressOpInsertMovieSegment
    callback MovieProgressProc (III-2105)
progressOpInsertTrackSegment
    callback MovieProgressProc (III-2105)
progressOpLoadMediaIntoRam
    callback MovieProgressProc (III-2105)
progressOpLoadMovieIntoRam
    callback MovieProgressProc (III-2105)
progressOpLoadTrackIntoRam
    callback MovieProgressProc (III-2105)

```

progressOpPaste	callback MovieProgressProc (III-2105)
progressProcAborted	“Error Codes” (IV-2663)
PropertyAtomAID	“Atom ID Codes” (IV-2649)
propertyNotSupportedByNodeErr	“Error Codes” (IV-2663)
'qdrq'	atom 'qdrq' (IV-2574)
qfcbNotCreatedErr	“Error Codes” (IV-2663)
qfcbNotFoundErr	“Error Codes” (IV-2663)
qtActionNotHandledErr	“Error Codes” (IV-2663)
qtcNeedsRateChanges	struct QTCallbackHeader (IV-2349)
qtcNeedsStartStopChanges	struct QTCallbackHeader (IV-2349)
qtcNeedsTimeChanges	struct QTCallbackHeader (IV-2349)
qtmID11EntryNotFoundErr	atom 'wtxt' (IV-2616)
qtmID11LoadErr	atom 'wtxt' (IV-2616)
qtmUninitialized	“Error Codes” (IV-2663)
qtNetworkAlreadyAllocatedErr	“Error Codes” (IV-2663)
qtParamErr	atom 'wtxt' (IV-2616)
qtvLibraryLoadErr	“Error Codes” (IV-2663)
qtvUninitialized	“Error Codes” (IV-2663)
quickTimeImageFileColorSyncProfileAtom	“Atom ID Codes” (IV-2649)
quickTimeImageFileImageDataAtom	“Atom ID Codes” (IV-2649)
quickTimeImageFileImageDescriptionAtom	“Atom ID Codes” (IV-2649)
quickTimeImageFileMetaDataAtom	“Atom ID Codes” (IV-2649)
quietCmd	“Sound Commands” (IV-2691)
rAliasType	atom 'wtxt' (IV-2616)
ramInit	struct GDevice (IV-2264)
rate11025hz	“Sound Sample Rates” (IV-2695)
rate11khz	“Sound Sample Rates” (IV-2695)
rate16khz	“Sound Sample Rates” (IV-2695)
rate22050hz	“Sound Sample Rates” (IV-2695)
rate22khz	“Sound Sample Rates” (IV-2695)
rate32khz	“Sound Sample Rates” (IV-2695)
rate44khz	“Sound Sample Rates” (IV-2695)
rate48khz	“Sound Sample Rates” (IV-2695)
rate8khz	“Sound Sample Rates” (IV-2695)
rateMultiplierCmd	“Sound Commands” (IV-2691)
'raw '	function SGGetSoundInputParameters (III-1733)
'rdrf'	atom 'rdrf' (IV-2575)
reallocPix	function NewImageGWorld (II-1066)
redColor	“Color Constants” (IV-2657)
ReferenceMovieAlternateGroupAID	“Atom ID Codes” (IV-2649)
ReferenceMovieComponentCheckAID	“Atom ID Codes” (IV-2649)
ReferenceMovieCPURatingAID	“Atom ID Codes” (IV-2649)
ReferenceMovieDataRateAID	“Atom ID Codes” (IV-2649)
ReferenceMovieDataRefAID	“Atom ID Codes” (IV-2649)
ReferenceMovieDescriptorAID	“Atom ID Codes” (IV-2649)
ReferenceMovieLanguageAID	“Atom ID Codes” (IV-2649)
ReferenceMovieQualityAID	“Atom ID Codes” (IV-2649)
ReferenceMovieRecordAID	“Atom ID Codes” (IV-2649)
ReferenceMovieVersionCheckAID	“Atom ID Codes” (IV-2649)

registerCmpGlobal	function RegisterComponent (III-1451)
registerCmpNoDuplicates	function RegisterComponent (III-1451)
registerCompAfter	function RegisterComponent (III-1451)
reInitCmd	“Sound Commands” (IV-2691)
REQUIRED_TEXT	function CreateShortcutMovieFile (I-149)
'reso'	atom 'reso' (IV-2576)
ResourceDataHandlerSubType	“Component Identifiers” (IV-2657)
resumeCmd	“Sound Commands” (IV-2691)
resumeFlag	struct EventRecord (IV-2246)
rgbComponentIn	function VDGetInputFormat (III-2005)
RgnClipAID	“Atom ID Codes” (IV-2649)
'rle'	function SGGetVideoCompressorType (III-1744)
'rmcd'	atom 'rmcd' (IV-2576)
'rmcs'	atom 'rmcs' (IV-2577)
'rmda'	atom 'rmda' (IV-2577)
'rmdr'	atom 'rmdr' (IV-2578)
'rmla'	atom 'rmla' (IV-2579)
'rmqu'	atom 'rmqu' (IV-2579)
'rmra'	atom 'rmra' (IV-2580)
'rmvc'	atom 'rmvc' (IV-2581)
'rpza'	function SGGetVideoCompressorType (III-1744)
sampldSynth	function SndNewChannel (III-1839)
samplesAlreadyInMediaErr	“Error Codes” (IV-2663)
SampleTableAID	“Atom ID Codes” (IV-2649)
scaleMatrixType	function GetMatrixType (I-419)
scaleTranslateMatrixType	function GetMatrixType (I-419)
scAllowZeroFrameRate	function SCGetInfo (III-1545)
scAllowZeroKeyFrameRate	function SCGetInfo (III-1545)
scCodecFlagsType	function SCGetInfo (III-1545)
scColorTableType	function SCGetInfo (III-1545)
scDataRateSettingsType	function SCGetInfo (III-1545)
scExtendedProcsType	function SCGetInfo (III-1545)
scheduledSoundCmd	“Sound Commands” (IV-2691)
scListEveryCodec	function SCGetInfo (III-1545)
scPreferCropping	function SCSetTestImagePictFile (III-1564)
scPreferenceFlagsType	function SCGetInfo (III-1545)
scPreferScaling	function SCSetTestImagePictFile (III-1564)
scPreferScalingAndCropping	function SCSetTestImagePictFile (III-1564)
scProgressProcType	function SCGetInfo (III-1545)
'scpt'	atom 'scpt' (IV-2581)
screenActive	struct GDevice (IV-2264)
screenDevice	struct GDevice (IV-2264)
scSequenceIDType	function SCGetInfo (III-1545)
scSettingsStateType	function SCGetInfo (III-1545)
scShowBestDepth	function SCGetInfo (III-1545)
scSoundChannelCountType	callback MovieExportGetPropertyProc (III-2102)
scSoundCompressionType	callback MovieExportGetPropertyProc (III-2102)
scSoundSampleRateType	callback MovieExportGetPropertyProc (III-2102)
scSoundSampleSizeType	callback MovieExportGetPropertyProc (III-2102)

scSpatialSettingsType	function SCGetInfo (III-1545)
scTemporalSettingsType	function SCGetInfo (III-1545)
scTypeNotFoundErr	"Error Codes" (IV-2663)
scUseMovableModal	function SCGetInfo (III-1545)
scWindowPositionType	function SCGetInfo (III-1545)
'sdp '	atom 'sdp ' (IV-2582)
'sean'	atom 'sean' (IV-2582)
secamIn	function VDGetVBlankRect (III-2021)
'seco'	atom 'wtxt' (IV-2616)
selectorNotSupportedByNodeErr	"Error Codes" (IV-2663)
'Sel0'	atom 'Sel0' (IV-2582)
seqGrabAppendToFile	function SGGetDataOutput (III-1711)
SeqGrabChannelType	"Component Identifiers" (IV-2657)
SeqGrabComponentType	"Component Identifiers" (IV-2657)
SeqGrabCompressionPanelType	"Component Identifiers" (IV-2657)
seqGrabDontAddMovieResource	function SGGetDataOutput (III-1711)
seqGrabDontMakeMovie	function SGGetDataOutput (III-1711)
seqGrabDontUseTempMemory	function SGGetDataOutput (III-1711)
seqGrabHasBounds	function SGGetChannelInfo (III-1702)
seqGrabHasDiscreteSamples	function SGGetChannelInfo (III-1702)
seqGrabHasVolume	function SGGetChannelInfo (III-1702)
SeqGrabPanelType	"Component Identifiers" (IV-2657)
seqGrabPause	function SGGetPause (III-1730)
seqGrabPauseForMenu	function SGGetPause (III-1730)
seqGrabPlayDuringRecord	function SGGetChannelUsage (III-1709)
seqGrabPreview	function SGGetChannelUsage (III-1709)
seqGrabRecord	function SGGetChannelUsage (III-1709)
seqGrabSettingsPreviewOnly	function SGSettingsDialog (III-1808)
SeqGrabSourcePanelType	"Component Identifiers" (IV-2657)
seqGrabToDisk	function SGGetDataOutput (III-1711)
seqGrabToMemory	function SGGetDataOutput (III-1711)
seqGrabUnpause	function SGGetPause (III-1730)
seqGrabWriteAppend	function SGAddExtendedMovieData (III-1678)
seqGrabWriteFill	function SGAddExtendedMovieData (III-1678)
seqGrabWriteReserve	function SGAddExtendedMovieData (III-1678)
settingNotSupportedByNodeErr	"Error Codes" (IV-2663)
sfItemBalloonHelp	callback DlHookProc (III-2079)
sfItemCancelButton	callback DlHookProc (III-2079)
sfItemDesktopButton	callback DlHookProc (III-2079)
sfItemDividerLinePict	callback DlHookProc (III-2079)
sfItemEjectButton	callback DlHookProc (III-2079)
sfItemFileListUser	callback DlHookProc (III-2079)
sfItemFileNameTextEdit	callback DlHookProc (III-2079)
sfItemNewFolderUser	callback DlHookProc (III-2079)
sfItemOpenButton	callback DlHookProc (III-2079)
sfItemPopUpMenuUser	callback DlHookProc (III-2079)
sfItemPromptStaticText	callback DlHookProc (III-2079)
sfItemVolumeUser	callback DlHookProc (III-2079)
sgDeviceListDontCheckAvailability	

	function SGGetChannelDeviceList (III-1701)
sgDeviceListWithIcons	function SGGetChannelDeviceList (III-1701)
sgDeviceNameFlagDeviceUnavailable	struct SGDeviceName (IV-2434)
sgFlagControlledGrab	function SGGetFlags (III-1718)
shiftKey	function QTVRMouseDown (II-1391)
ShowFilePreviewComponentType	"Component Identifiers" (IV-2657)
showUserSettingsDialog	function ConvertFileToMovieFile (I-129)
siActiveChannels	"Sound Information Selectors" (IV-2693)
siActiveLevels	"Sound Information Selectors" (IV-2693)
siAGCOnOff	"Sound Information Selectors" (IV-2693)
siAsync	"Sound Information Selectors" (IV-2693)
siAVDisplayBehavior	"Sound Information Selectors" (IV-2693)
siBestQuality	function SndRecord (III-1843)
siBetterQuality	function SndRecord (III-1843)
siChannelAvailable	"Sound Information Selectors" (IV-2693)
siCompressionAvailable	"Sound Information Selectors" (IV-2693)
siCompressionChannels	"Sound Information Selectors" (IV-2693)
siCompressionFactor	"Sound Information Selectors" (IV-2693)
siCompressionHeader	"Sound Information Selectors" (IV-2693)
siCompressionNames	"Sound Information Selectors" (IV-2693)
siCompressionParams	"Sound Information Selectors" (IV-2693)
siCompressionSampleRate	"Sound Information Selectors" (IV-2693)
siCompressionType	"Sound Information Selectors" (IV-2693)
siContinuous	"Sound Information Selectors" (IV-2693)
siDecompressionParams	"Sound Information Selectors" (IV-2693)
siDeviceBufferInfo	"Sound Information Selectors" (IV-2693)
siDeviceConnected	"Sound Information Selectors" (IV-2693)
siDeviceIcon	"Sound Information Selectors" (IV-2693)
siDeviceName	"Sound Information Selectors" (IV-2693)
siEQSpectrumBands	"Sound Information Selectors" (IV-2693)
siEQSpectrumLevels	"Sound Information Selectors" (IV-2693)
siEQSpectrumOnOff	"Sound Information Selectors" (IV-2693)
siEQToneControlGain	"Sound Information Selectors" (IV-2693)
siEQToneControlOnOff	"Sound Information Selectors" (IV-2693)
sigAdobePremiere	"File Types and Creators" (IV-2668)
sigAfterDark	"File Types and Creators" (IV-2668)
sigAldusSuper3D	"File Types and Creators" (IV-2668)
sigAutoCAD	"File Types and Creators" (IV-2668)
sigClip3D	"File Types and Creators" (IV-2668)
sigColorMacCheese	"File Types and Creators" (IV-2668)
sigCricketDraw	"File Types and Creators" (IV-2668)
sigCricketGraph	"File Types and Creators" (IV-2668)
sigDeltagraphPro	"File Types and Creators" (IV-2668)
sigDesign2	"File Types and Creators" (IV-2668)
sigDesignCAD	"File Types and Creators" (IV-2668)
sigDesignStudio	"File Types and Creators" (IV-2668)
sigDigDarkroom	"File Types and Creators" (IV-2668)
sigDreams	"File Types and Creators" (IV-2668)
sigDynaperspective	"File Types and Creators" (IV-2668)

sigGenericCADD	"File Types and Creators" (IV-2668)
sigGraphMaster	"File Types and Creators" (IV-2668)
sigImageStudio	"File Types and Creators" (IV-2668)
sigInfiniD	"File Types and Creators" (IV-2668)
sigKaleidaGraph	"File Types and Creators" (IV-2668)
sigKidPix	"File Types and Creators" (IV-2668)
sigLabVIEW	"File Types and Creators" (IV-2668)
sigMacDraft	"File Types and Creators" (IV-2668)
sigMacDraw	"File Types and Creators" (IV-2668)
sigMacFlow	"File Types and Creators" (IV-2668)
sigMacSpin	"File Types and Creators" (IV-2668)
sigMiniCad	"File Types and Creators" (IV-2668)
sigModelShop	"File Types and Creators" (IV-2668)
sigMoviePlayer	"File Types and Creators" (IV-2668)
sigMovieRecorder	"File Types and Creators" (IV-2668)
sigOasis	"File Types and Creators" (IV-2668)
sigOBJECTMASTER	"File Types and Creators" (IV-2668)
sigOfoto	"File Types and Creators" (IV-2668)
sigOmnis5	"File Types and Creators" (IV-2668)
siGoodQuality	function SndRecord (III-1843)
sigOptix	"File Types and Creators" (IV-2668)
sigPhotoMac	"File Types and Creators" (IV-2668)
sigPictureCompressor	"File Types and Creators" (IV-2668)
sigPICTViewer	"File Types and Creators" (IV-2668)
sigPixelPaint	"File Types and Creators" (IV-2668)
sigScreenPlay	"File Types and Creators" (IV-2668)
sigSmoothie	"File Types and Creators" (IV-2668)
sigStudio1	"File Types and Creators" (IV-2668)
sigStudio32	"File Types and Creators" (IV-2668)
sigStudio8	"File Types and Creators" (IV-2668)
sigSwivel3D	"File Types and Creators" (IV-2668)
sigVersaCad	"File Types and Creators" (IV-2668)
siHardwareBalance	"Sound Information Selectors" (IV-2693)
siHardwareBalanceSteps	"Sound Information Selectors" (IV-2693)
siHardwareBass	"Sound Information Selectors" (IV-2693)
siHardwareBassSteps	"Sound Information Selectors" (IV-2693)
siHardwareBusy	"Sound Information Selectors" (IV-2693)
siHardwareFormat	"Sound Information Selectors" (IV-2693)
siHardwareMute	"Sound Information Selectors" (IV-2693)
siHardwareTreble	"Sound Information Selectors" (IV-2693)
siHardwareTrebleSteps	"Sound Information Selectors" (IV-2693)
siHardwareVolume	"Sound Information Selectors" (IV-2693)
siHardwareVolumeSteps	"Sound Information Selectors" (IV-2693)
siHeadphoneMute	"Sound Information Selectors" (IV-2693)
siHeadphoneVolume	"Sound Information Selectors" (IV-2693)
siHeadphoneVolumeSteps	"Sound Information Selectors" (IV-2693)
siInputAvailable	"Sound Information Selectors" (IV-2693)
siInputGain	"Sound Information Selectors" (IV-2693)
siInputSource	"Sound Information Selectors" (IV-2693)

siInputSourceNames	"Sound Information Selectors" (IV-2693)
siLevelMeterOnOff	"Sound Information Selectors" (IV-2693)
siModemGain	"Sound Information Selectors" (IV-2693)
siMonitorAvailable	"Sound Information Selectors" (IV-2693)
siMonitorSource	"Sound Information Selectors" (IV-2693)
siNumberChannels	"Sound Information Selectors" (IV-2693)
siOptionsDialog	"Sound Information Selectors" (IV-2693)
siOSTypeInputAvailable	"Sound Information Selectors" (IV-2693)
siOSTypeInputSource	"Sound Information Selectors" (IV-2693)
siPlayThruOnOff	"Sound Information Selectors" (IV-2693)
siPostMixerSoundComponent	"Sound Information Selectors" (IV-2693)
siPreMixerSoundComponent	"Sound Information Selectors" (IV-2693)
siQuality	"Sound Information Selectors" (IV-2693)
siRateMultiplier	"Sound Information Selectors" (IV-2693)
siReadPermission	function SPBOpenDevice (III-1876)
siRecordingQuality	"Sound Information Selectors" (IV-2693)
siSampleRate	"Sound Information Selectors" (IV-2693)
siSampleRateAvailable	"Sound Information Selectors" (IV-2693)
siSampleSize	"Sound Information Selectors" (IV-2693)
siSampleSizeAvailable	"Sound Information Selectors" (IV-2693)
siSetupCDAudio	"Sound Information Selectors" (IV-2693)
siSetupModemAudio	"Sound Information Selectors" (IV-2693)
siSlopeAndIntercept	"Sound Information Selectors" (IV-2693)
siSoundClock	"Sound Information Selectors" (IV-2693)
siSpeakerMute	"Sound Information Selectors" (IV-2693)
siSpeakerVolume	"Sound Information Selectors" (IV-2693)
siSSpCPULoadLimit	"Sound Information Selectors" (IV-2693)
siSSpLocalization	"Sound Information Selectors" (IV-2693)
siSSpSpeakerSetup	"Sound Information Selectors" (IV-2693)
siStereoInputGain	"Sound Information Selectors" (IV-2693)
siSubwooferMute	"Sound Information Selectors" (IV-2693)
siTwosComplementOnOff	"Sound Information Selectors" (IV-2693)
siUseThisSoundClock	"Sound Information Selectors" (IV-2693)
siVolume	"Sound Information Selectors" (IV-2693)
siVoxRecordInfo	"Sound Information Selectors" (IV-2693)
siVoxStopInfo	"Sound Information Selectors" (IV-2693)
siWideStereo	"Sound Information Selectors" (IV-2693)
siWritePermission	function SPBOpenDevice (III-1876)
sixToOne	struct CmpSoundHeader (IV-2176)
sixToOnePacketSize	struct CmpSoundHeader (IV-2176)
'skip'	atom 'skip' (IV-2583)
SkipAtomType	"Atom ID Codes" (IV-2649)
smArabic	"Localization Codes" (IV-2673)
smArmenian	"Localization Codes" (IV-2673)
smBengali	"Localization Codes" (IV-2673)
smBurmese	"Localization Codes" (IV-2673)
'smc '	function SGGetVideoCompressorType (III-1744)
smCentralEuroRoman	atom 'wtxt' (IV-2616)
smCurrentScript	function FlattenMovieData (I-374)

smCyrillic	"Localization Codes" (IV-2673)
smDevanagari	"Localization Codes" (IV-2673)
smEthiopic	"Localization Codes" (IV-2673)
smExtArabic	"Localization Codes" (IV-2673)
smGeez	"Localization Codes" (IV-2673)
smGeorgian	"Localization Codes" (IV-2673)
smGreek	"Localization Codes" (IV-2673)
smGujarati	"Localization Codes" (IV-2673)
smGurmukhi	"Localization Codes" (IV-2673)
'smhd'	atom 'smhd' (IV-2584)
smHebrew	"Localization Codes" (IV-2673)
smJapanese	"Localization Codes" (IV-2673)
smKannada	"Localization Codes" (IV-2673)
smKhmer	"Localization Codes" (IV-2673)
smKorean	"Localization Codes" (IV-2673)
smLao	"Localization Codes" (IV-2673)
smMalayalam	"Localization Codes" (IV-2673)
smMongolian	"Localization Codes" (IV-2673)
smOriya	"Localization Codes" (IV-2673)
smRoman	"Localization Codes" (IV-2673)
smRSymbol	atom 'wtxt' (IV-2616)
smSimpChinese	"Localization Codes" (IV-2673)
smSinhalese	"Localization Codes" (IV-2673)
smSystemScript	function FlattenMovieData (I-374)
smTamil	"Localization Codes" (IV-2673)
smTelugu	"Localization Codes" (IV-2673)
smThai	"Localization Codes" (IV-2673)
smTibetan	"Localization Codes" (IV-2673)
smTradChinese	"Localization Codes" (IV-2673)
smUnicodeScript	"Localization Codes" (IV-2673)
smUninterp	"Localization Codes" (IV-2673)
smVietnamese	"Localization Codes" (IV-2673)
soundCmd	"Sound Commands" (IV-2691)
SoundLocalizationAID	"Atom ID Codes" (IV-2649)
SoundMediaInfoHeaderAID	"Atom ID Codes" (IV-2649)
SoundMediaType	function GetMediaHandlerDescription (I-432)
soundSupportNotAvailableErr	"Error Codes" (IV-2663)
sourceNotFoundErr	"Error Codes" (IV-2663)
spriteHitTestBounds	function SpriteHitTest (III-1885)
spriteHitTestImage	function SpriteHitTest (III-1885)
spriteHitTestInvisibleSprites	function SpriteHitTest (III-1885)
spriteHitTestIsClick	function SpriteHitTest (III-1885)
spriteHitTestLocInDisplayCoordinates	function SpriteHitTest (III-1885)
SpriteMediaType	"Component Identifiers" (IV-2657)
'sprt'	atom 'sprt' (IV-2584)
'sptl'	atom 'sptl' (IV-2586)
squareWaveSynth	function SndNewChannel (III-1839)
srcBic	struct CGrafPort (IV-2168)
srcBic,	atom 'wtxt' (IV-2616)

srcCopy	"Graphics Transfer Modes" (IV-2670)
srcCopy,	atom 'wtxt' (IV-2616)
srcOr	struct CGrafPort (IV-2168)
srcOr,	atom 'wtxt' (IV-2616)
srcXor	struct CGrafPort (IV-2168)
srcXor,	atom 'wtxt' (IV-2616)
'ssrc'	atom 'ssrc' (IV-2586)
staleEditState	"Error Codes" (IV-2663)
StandardCompressionSubType	"Component Identifiers" (IV-2657)
StandardCompressionSubTypeSound	"Component Identifiers" (IV-2657)
StandardCompressionType	"Component Identifiers" (IV-2657)
stateBlockSize	struct StateBlock (IV-2463)
'stbl'	atom 'stbl' (IV-2587)
STChunkOffset64AID	"Atom ID Codes" (IV-2649)
STChunkOffsetAID	"Atom ID Codes" (IV-2649)
'stco'	atom 'stco' (IV-2589)
stdSH	struct CmpSoundHeader (IV-2176)
'STR '	struct ResourceSpec (IV-2402)
streamingNodeNotReadyErr	"Error Codes" (IV-2663)
stretchPix	function NewImageGWorld (II-1066)
'strt'	atom 'strt' (IV-2589)
STSampleDescAID	"Atom ID Codes" (IV-2649)
STSampleIDAID	"Atom ID Codes" (IV-2649)
STSampleSizeAID	"Atom ID Codes" (IV-2649)
STSampleToChunkAID	"Atom ID Codes" (IV-2649)
'stsc'	atom 'stsc' (IV-2590)
'stds'	atom 'stds' (IV-2591)
'stsh'	atom 'stsh' (IV-2592)
STShadowSyncAID	"Atom ID Codes" (IV-2649)
'stss'	atom 'stss' (IV-2593)
STSyncSampleAID	"Atom ID Codes" (IV-2649)
'stsz'	atom 'stsz' (IV-2594)
STTimeToSampAID	"Atom ID Codes" (IV-2649)
'stts'	atom 'stts' (IV-2595)
subOver	function SetDSequenceTransferMode (III-1591)
subPin	function SetDSequenceTransferMode (III-1591)
suppressDither	function DrawTrimmedPicture (I-311)
suspendResumeMessage	struct EventRecord (IV-2246)
sVideoIn	function VDGetInputFormat (III-2005)
'sync'	atom 'sync' (IV-2596)
syncCmd	"Sound Commands" (IV-2691)
synthesizerNotRespondingOSErr	"Error Codes" (IV-2663)
synthesizerOSErr	"Error Codes" (IV-2663)
sysBeepDisable	function SndGetSysBeepState (III-1833)
sysBeepEnable	function SndGetSysBeepState (III-1833)
systemMicrosecondClock	"Component Identifiers" (IV-2657)
systemMillisecondClock	"Component Identifiers" (IV-2657)
systemSecondClock	"Component Identifiers" (IV-2657)
systemTickClock	"Component Identifiers" (IV-2657)

'tbox'	atom 'tbox' (IV-2597)
tc24HourMax	struct TimeCodeDef (IV-2482)
tcCounter	struct TimeCodeDef (IV-2482)
tcdfShowTimeCode	function TCGetTimeCodeFlags (III-1921)
tcDropFrame	struct TimeCodeDef (IV-2482)
'tcmi'	atom 'tcmi' (IV-2597)
tcNegTimesOK	struct TimeCodeDef (IV-2482)
tctNegFlag	struct TimeCodeTime (IV-2485)
teCenter	function SGSetJustification (III-1795)
teFlushDefault	function SGSetJustification (III-1795)
teFlushLeft	function SGSetJustification (III-1795)
teFlushRight	function SGSetJustification (III-1795)
TextMediaType	function GetMediaHandlerDescription (I-432)
ThreeDeeMediaType	"Component Identifiers" (IV-2657)
threeToOne	struct CmpSoundHeader (IV-2176)
threeToOnePacketSize	struct CmpSoundHeader (IV-2176)
'tick'	atom 'wtxt' (IV-2616)
timeBaseAfterStopTime	function GetTimeBaseStatus (I-519)
timeBaseBeforeStartTime	function GetTimeBaseStatus (I-519)
TimeCodeMediaType	"Component Identifiers" (IV-2657)
timeNotInMedia	"Error Codes" (IV-2663)
timeNotInTrack	"Error Codes" (IV-2663)
timeNotInViewErr	"Error Codes" (IV-2663)
'tkhd'	atom 'tkhd' (IV-2598)
'tmax'	atom 'tmax' (IV-2598)
'tmcd'	atom 'tmcd' (IV-2599)
'tmin'	atom 'tmin' (IV-2599)
'tpyl'	atom 'tpyl' (IV-2600)
TrackAID	"Atom ID Codes" (IV-2649)
TrackEnable	struct TrackHeader (IV-2489)
TrackHeaderAID	"Atom ID Codes" (IV-2649)
trackIDNotFound	"Error Codes" (IV-2663)
TrackInMovie	struct TrackHeader (IV-2489)
TrackInPoster	struct TrackHeader (IV-2489)
TrackInPreview	struct TrackHeader (IV-2489)
trackNotInMovie	"Error Codes" (IV-2663)
TrackReferenceAID	"Atom ID Codes" (IV-2649)
trackUsageInMovie	function GetTrackUsage (I-545)
trackUsageInPoster	function GetTrackUsage (I-545)
trackUsageInPreview	function GetTrackUsage (I-545)
'trak'	atom 'trak' (IV-2600)
translateMatrixType	function GetMatrixType (I-419)
transparent	function SetDSequenceTransferMode (III-1591)
'tref'	atom 'tref' (IV-2602)
triggerAtStart	function NewCallback (II-1053)
triggerAtStop	function NewCallback (II-1053)
triggerRateChange	function CallMeWhen (I-67)
triggerRateEqual	function CallMeWhen (I-67)
triggerRateGT	function CallMeWhen (I-67)

triggerRateGTE	function CallMeWhen (I-67)
triggerRateLT	function CallMeWhen (I-67)
triggerRateLTE	function CallMeWhen (I-67)
triggerRateNotEqual	function CallMeWhen (I-67)
triggerTimeBwd	function CallMeWhen (I-67)
triggerTimeEither	function CallMeWhen (I-67)
triggerTimeFwd	function CallMeWhen (I-67)
'trpy'	atom 'trpy' (IV-2603)
tuneParseOSError	"Error Codes" (IV-2663)
tunePlayerFullOSError	"Error Codes" (IV-2663)
'twdt'	atom 'twdt' (IV-2604)
'twdu'	atom 'twdu' (IV-2604)
TweenMediaType	"Component Identifiers" (IV-2657)
'twen'	atom 'twen' (IV-2605)
'twnt'	atom 'twnt' (IV-2605)
'twos'	struct InstSampleDescRec (IV-2296)
'twst'	atom 'twst' (IV-2606)
' ty'	atom ' ty' (IV-2606)
'udta'	atom 'udta' (IV-2607)
unkeepInRam	function LoadMediaIntoRam (II-779)
unknownFormatErr	"Error Codes" (IV-2663)
unsupportedAuxiliaryImportData	"Error Codes" (IV-2663)
unsupportedOSError	"Error Codes" (IV-2663)
unsupportedProcessorErr	"Error Codes" (IV-2663)
unused1	function SPBRecord (III-1878)
updateEvt	struct EventRecord (IV-2246)
URLDataHandlerSubType	"Component Identifiers" (IV-2657)
urlDataHFTPBadNameListErr	"Error Codes" (IV-2663)
urlDataHFTPBadPasswordErr	"Error Codes" (IV-2663)
urlDataHFTPBadUserErr	"Error Codes" (IV-2663)
urlDataHFTPDataConnectionErr	"Error Codes" (IV-2663)
urlDataHFTPFilenameErr	"Error Codes" (IV-2663)
urlDataHFTPNeedPasswordErr	"Error Codes" (IV-2663)
urlDataHFTPNoDirectoryErr	"Error Codes" (IV-2663)
urlDataHFTPNoNetDriverErr	"Error Codes" (IV-2663)
urlDataHFTPNoPasswordErr	"Error Codes" (IV-2663)
urlDataHFTPPermissionsErr	"Error Codes" (IV-2663)
urlDataHFTPProtocolErr	"Error Codes" (IV-2663)
urlDataHFTPQuotaErr	"Error Codes" (IV-2663)
urlDataHFTPServerDisconnectedErr	"Error Codes" (IV-2663)
urlDataHFTPServerError	"Error Codes" (IV-2663)
urlDataHFTPShutdownErr	"Error Codes" (IV-2663)
urlDataHFTPURLErr	"Error Codes" (IV-2663)
urlDataHHTTNoNetDriverErr	"Error Codes" (IV-2663)
urlDataHHTTProtocolErr	"Error Codes" (IV-2663)
urlDataHHTTRedirectErr	"Error Codes" (IV-2663)
urlDataHHTTURLErr	"Error Codes" (IV-2663)
userCanceledErr	function QTIsStandardParameterDialogEvent (II-1186)
UserDataAID	"Atom ID Codes" (IV-2649)

userDataItemNotFound	"Error Codes" (IV-2663)
userLong	function SPBRecord (III-1878)
useTempMem	function NewImageGWorld (II-1066)
variableCompression	function GetCompressionInfo (I-398)
vdTypeAlpha	struct DigitizerInfo (IV-2234)
vdTypeBasic	struct DigitizerInfo (IV-2234)
vdTypeKey	struct DigitizerInfo (IV-2234)
vdTypeMask	struct DigitizerInfo (IV-2234)
vdUseAnyField	function VDGetFieldPreference (III-2001)
vdUseEvenField	function VDGetFieldPreference (III-2001)
vdUseOddField	function VDGetFieldPreference (III-2001)
verAfrikaans	"Localization Codes" (IV-2673)
verArabic	"Localization Codes" (IV-2673)
verArmenian	"Localization Codes" (IV-2673)
verAustralia	"Localization Codes" (IV-2673)
verAustria	"Localization Codes" (IV-2673)
verBengali	"Localization Codes" (IV-2673)
verBhutan	"Localization Codes" (IV-2673)
verBrazil	"Localization Codes" (IV-2673)
verBreton	"Localization Codes" (IV-2673)
verBritain	"Localization Codes" (IV-2673)
verBulgaria	"Localization Codes" (IV-2673)
verByeloRussian	"Localization Codes" (IV-2673)
verCatalonia	"Localization Codes" (IV-2673)
verChina	"Localization Codes" (IV-2673)
verCroatia	"Localization Codes" (IV-2673)
verCyprus	"Localization Codes" (IV-2673)
verCzech	"Localization Codes" (IV-2673)
verDenmark	"Localization Codes" (IV-2673)
verEngCanada	"Localization Codes" (IV-2673)
verEsperanto	"Localization Codes" (IV-2673)
verEstonia	"Localization Codes" (IV-2673)
verFarEastGeneric	atom 'wtxt' (IV-2616)
verFaroeIsl	"Localization Codes" (IV-2673)
verFinland	"Localization Codes" (IV-2673)
verFlemish	"Localization Codes" (IV-2673)
verFrance	"Localization Codes" (IV-2673)
verFrBelgium	"Localization Codes" (IV-2673)
verFrCanada	"Localization Codes" (IV-2673)
verFrenchUniversal	"Localization Codes" (IV-2673)
verFrSwiss	"Localization Codes" (IV-2673)
verGeorgian	"Localization Codes" (IV-2673)
verGermany	"Localization Codes" (IV-2673)
verGreece	atom 'wtxt' (IV-2616)
verGreecePoly	atom 'wtxt' (IV-2616)
verGreenland	"Localization Codes" (IV-2673)
verGrSwiss	"Localization Codes" (IV-2673)
verGujarati	"Localization Codes" (IV-2673)
verHungary	"Localization Codes" (IV-2673)

verIceland	"Localization Codes" (IV-2673)
verIndiaHindi	"Localization Codes" (IV-2673)
verIndiaUrdu	"Localization Codes" (IV-2673)
verInternational	atom 'wtxt' (IV-2616)
verIran	"Localization Codes" (IV-2673)
verIreland	"Localization Codes" (IV-2673)
verIrishGaelicScript	"Localization Codes" (IV-2673)
verIsrael	"Localization Codes" (IV-2673)
verItalianSwiss	"Localization Codes" (IV-2673)
verItaly	"Localization Codes" (IV-2673)
verJapan	"Localization Codes" (IV-2673)
verKorea	"Localization Codes" (IV-2673)
verLatvia	"Localization Codes" (IV-2673)
verLithuania	"Localization Codes" (IV-2673)
verMacedonian	"Localization Codes" (IV-2673)
verMagyar	"Localization Codes" (IV-2673)
verMalta	"Localization Codes" (IV-2673)
verManxGaelic	"Localization Codes" (IV-2673)
verMarathi	"Localization Codes" (IV-2673)
verMultilingual	atom 'wtxt' (IV-2616)
verNepal	"Localization Codes" (IV-2673)
verNetherlands	"Localization Codes" (IV-2673)
verNorway	"Localization Codes" (IV-2673)
verNunavut	"Localization Codes" (IV-2673)
verNynorsk	"Localization Codes" (IV-2673)
verPakistanUrdu	"Localization Codes" (IV-2673)
verPoland	"Localization Codes" (IV-2673)
verPortugal	"Localization Codes" (IV-2673)
verPunjabi	"Localization Codes" (IV-2673)
verRomania	"Localization Codes" (IV-2673)
verRussia	"Localization Codes" (IV-2673)
verSami	"Localization Codes" (IV-2673)
verScottishGaelic	"Localization Codes" (IV-2673)
verScriptGeneric	atom 'wtxt' (IV-2616)
verSerbian	"Localization Codes" (IV-2673)
verSingapore	"Localization Codes" (IV-2673)
versionCmd	"Sound Commands" (IV-2691)
verSlovak	"Localization Codes" (IV-2673)
verSlovenian	"Localization Codes" (IV-2673)
verSpain	atom 'wtxt' (IV-2616)
verSplatinAmerica	atom 'wtxt' (IV-2616)
verSweden	"Localization Codes" (IV-2673)
verTaiwan	"Localization Codes" (IV-2673)
verThailand	"Localization Codes" (IV-2673)
verTibetan	"Localization Codes" (IV-2673)
verTonga	"Localization Codes" (IV-2673)
verTurkey	"Localization Codes" (IV-2673)
verTurkishModified	"Localization Codes" (IV-2673)
verUkraine	"Localization Codes" (IV-2673)

verUS	"Localization Codes" (IV-2673)
verUzbek	"Localization Codes" (IV-2673)
verVietnam	"Localization Codes" (IV-2673)
verWelsh	"Localization Codes" (IV-2673)
'vide'	atom 'vide' (IV-2610)
videoDigitizerComponentType	"Component Identifiers" (IV-2657)
VideoMediaInfoHeaderAID	"Atom ID Codes" (IV-2649)
VideoMediaType	function GetMediaHandlerDescription (I-432)
videoOutputInUseErr	"Error Codes" (IV-2663)
VisualMediaCharacteristic	function GetMovieNextInterestingTime (I-480)
'vmhd'(media)	atom 'vmhd'(media) (IV-2611)
'vmhd'(transfer)	atom 'vmhd'(transfer) (IV-2612)
volumeCmd	"Sound Commands" (IV-2691)
'vrcp'	atom 'vrcp' (IV-2612)
'vrni'	atom 'vrni' (IV-2613)
'vrnp'	atom 'vrnp' (IV-2614)
'vrsc'	atom 'vrsc' (IV-2614)
wackBadFileErr	"Error Codes" (IV-2663)
wackBadMetaDataErr	"Error Codes" (IV-2663)
wackForkNotFoundErr	"Error Codes" (IV-2663)
waitCmd	"Sound Commands" (IV-2691)
waveTableSynth	function SndNewChannel (III-1839)
wfFileNotFound	"Error Codes" (IV-2663)
whiteColor	"Color Constants" (IV-2657)
'wide'	atom 'wide' (IV-2614)
WideAtomPlaceholderType	"Atom ID Codes" (IV-2649)
WiredActionHandlerType	"Component Identifiers" (IV-2657)
'WLOC'	atom 'WLOC' (IV-2615)
'wtxt'	atom 'wtxt' (IV-2616)
yellowColor	"Color Constants" (IV-2657)
zlibDataCompressorSubType	atom 'dcom' (IV-2527)

Functions By Header Files

QuickTime Header Files

Components.h	IV-3005
Endian.h	IV-3007
ImageCodec.h	IV-3007
ImageCompression.h	IV-3009
MediaHandlers.h	IV-3016
Movies.h	IV-3017
MoviesFormat.h	IV-3029
QTML.h	IV-3029
QTSMovie.h	IV-3030
QTStreamingComponents.h	IV-3030
QuickTimeComponents.h	IV-3032
QuickTimeMusic.h	IV-3041
QuickTimeStreaming.h	IV-3044
QuickTimeVR.h	IV-3046
QuickTimeVRFormat.h	IV-3048
Sound.h	IV-3048

Components.h

NewComponentFunctionUPP	II-1056
DisposeComponentFunctionUPP	I-258
RegisterComponent	III-1451
RegisterComponentResource	III-1453
UnregisterComponent	III-1979
FindNextComponent	I-360
CountComponents	I-143
GetComponentInfo	I-389
GetComponentListModSeed	I-391
GetComponentTypeModSeed	I-395
OpenAComponent	II-1126
OpenComponent	II-1130
CloseComponent	I-100
GetComponentInstanceError	I-390
SetComponentInstanceError	III-1571

GetComponentRefcon I-394
SetComponentRefcon III-1573
OpenComponentResFile II-1130
OpenAComponentResFile II-1127
CloseComponentResFile I-101
GetComponentResource I-394
GetComponentIndString I-388
ResolveComponentAlias III-1462
GetComponentPublicResource I-392
GetComponentPublicResourceList I-393
GetComponentInstanceStorage I-391
SetComponentInstanceStorage III-1572
GetComponentInstanceA5 I-390
SetComponentInstanceA5 III-1570
CountComponentInstances I-142
CallComponentFunction I-61
CallComponentFunctionWithStorage I-61
CallComponentFunctionWithStorageProcInfo I-62
DelegateComponentCall I-249
SetDefaultComponent III-1581
OpenDefaultComponent II-1131
OpenADefaultComponent II-1129
CaptureComponent I-74
UncaptureComponent III-1979
RegisterComponentResourceFile III-1455
GetComponentIconSuite I-387
ComponentFunctionImplemented I-111
GetComponentVersion I-395
ComponentSetTarget I-112
CallComponentOpen I-65
CallComponentClose I-59
CallComponentCanDo I-58
CallComponentVersion I-67
CallComponentRegister I-65
CallComponentTarget I-66
CallComponentUnregister I-66
CallComponentGetMPWorkFunction I-63
CallComponentGetPublicResource I-64
CallComponent I-58
CallComponentDispatch I-59
NewComponentMPWorkFunctionUPP II-1056
NewComponentRoutineUPP II-1057
NewGetMissingComponentResourceUPP II-1059
DisposeComponentMPWorkFunctionUPP I-258
DisposeComponentRoutineUPP I-259
DisposeGetMissingComponentResourceUPP I-261

Endian.h

EndianS16_BtoN I-317
 EndianS16_NtoB I-319
 EndianS16_LtoN I-318
 EndianS16_NtoL I-320
 EndianS16_LtoB I-318
 EndianS16_BtoL I-317
 EndianU16_BtoN I-326
 EndianU16_NtoB I-327
 EndianU16_LtoN I-327
 EndianU16_NtoL I-328
 EndianU16_LtoB I-326
 EndianU16_BtoL I-326
 EndianS32_BtoN I-321
 EndianS32_NtoB I-322
 EndianS32_LtoN I-322
 EndianS32_NtoL I-323
 EndianS32_LtoB I-321
 EndianS32_BtoL I-321
 EndianU32_BtoN I-329
 EndianU32_NtoB I-331
 EndianU32_LtoN I-330
 EndianU32_NtoL I-332
 EndianU32_LtoB I-330
 EndianU32_BtoL I-329
 EndianS64_BtoN I-324
 EndianS64_NtoB I-325
 EndianS64_LtoN I-324
 EndianS64_NtoL I-325
 EndianS64_LtoB I-324
 EndianS64_BtoL I-323
 EndianU64_BtoN I-333
 EndianU64_NtoB I-334
 EndianU64_LtoN I-334
 EndianU64_NtoL I-334
 EndianU64_LtoB I-333
 EndianU64_BtoL I-332

ImageCodec.h

NewImageCodecTimeTriggerUPP II-1065
 DisposeImageCodecTimeTriggerUPP I-267
 NewImageCodecMPDrawBandUPP II-1065
 DisposeImageCodecMPDrawBandUPP I-266
 ImageCodecGetCodecInfo II-710
 ImageCodecGetCompressionTime II-711

ImageCodecGetMaxCompressionSize II-714
ImageCodecPreCompress II-730
ImageCodecBandCompress II-688
ImageCodecPreDecompress II-731
ImageCodecBandDecompress II-689
ImageCodecBusy II-691
ImageCodecGetCompressedImageSize II-710
ImageCodecGetSimilarity II-720
ImageCodecTrimImage II-743
ImageCodecRequestSettings II-735
ImageCodecGetSettings II-719
ImageCodecSetSettings II-737
ImageCodecFlush II-708
ImageCodecSetTimeCode II-738
ImageCodecIsImageDescriptionEquivalent II-725
ImageCodecNewMemory II-729
ImageCodecDisposeMemory II-696
ImageCodecHitTestData II-722
ImageCodecNewImageBufferMemory II-727
ImageCodecExtractAndCombineFields II-705
ImageCodecGetMaxCompressionSizeWithSources II-716
ImageCodecSetTimeBase II-738
ImageCodecSourceChanged II-739
ImageCodecFlushFrame II-708
ImageCodecGetSettingsAsText II-720
ImageCodecGetParameterListHandle II-718
ImageCodecGetParameterList II-717
ImageCodecCreateStandardParameterDialog II-692
ImageCodecIsStandardParameterDialogEvent II-726
ImageCodecDismissStandardParameterDialog II-695
ImageCodecStandardParameterDialogDoAction II-740
ImageCodecNewImageGWorld II-728
ImageCodecDisposeImageGWorld II-696
ImageCodecHitTestDataWithFlags II-723
ImageCodecValidateParameters II-745
ImageCodecGetBaseMPWorkFunction II-709
ImageCodecPreflight II-732
ImageCodecInitialize II-724
ImageCodecBeginBand II-690
ImageCodecDrawBand II-697
ImageCodecEndBand II-704
ImageCodecQueueStarting II-733
ImageCodecQueueStopping II-734
ImageCodecDroppingFrame II-698
ImageCodecScheduleFrame II-736
ImageCodecCancelTrigger II-692
QTPhotoSetSampling II-1213
QTPhotoSetRestartInterval II-1213
QTPhotoDefineHuffmanTable II-1211

QTPhotoDefineQuantizationTable II-1212
 ImageCodecEffectSetup II-704
 ImageCodecEffectBegin II-698
 ImageCodecEffectRenderFrame II-702
 ImageCodecEffectConvertEffectSourceToFormat II-699
 ImageCodecEffectCancel II-699
 ImageCodecEffectGetSpeed II-700
 ImageCodecEffectPrepareSMPTEFrame II-701
 ImageCodecEffectDisposeSMPTEFrame II-700
 ImageCodecEffectRenderSMPTEFrame II-702
 CurveGetLength I-155
 CurveLengthToPoint I-159
 CurveNewPath I-160
 CurveCountPointsInPath I-153
 CurveGetPathPoint I-157
 CurveInsertPointIntoPath I-158
 CurveSetPathPoint I-162
 CurveGetNearestPathPoint I-156
 CurvePathPointToLength I-161
 CurveCreateVectorStream I-154
 CurveAddAtomToVectorStream I-151
 CurveAddPathAtomToVectorStream I-152
 CurveAddZeroAtomToVectorStream I-152
 CurveGetAtomDataFromVectorStream I-154

ImageCompression.h

NewICMDataUPP II-1062
 NewICMFlushUPP II-1063
 NewICMCompletionUPP II-1061
 NewICMProgressUPP II-1064
 NewStdPixUPP II-1118
 NewQDPixUPP II-1096
 NewICMAlignmentUPP II-1060
 NewICMCursorShieldedUPP II-1062
 NewICMMemoryDisposedUPP II-1063
 NewICMConvertDataFormatUPP II-1061
 DisposeICMDataUPP I-264
 DisposeICMFlushUPP I-264
 DisposeICMCompletionUPP I-263
 DisposeICMProgressUPP I-265
 DisposeStdPixUPP I-299
 DisposeQDPixUPP I-281
 DisposeICMAlignmentUPP I-262
 DisposeICMCursorShieldedUPP I-264
 DisposeICMMemoryDisposedUPP I-265
 DisposeICMConvertDataFormatUPP I-263
 CodecManagerVersion I-103

GetCodecNameList I-386
DisposeCodecNameList I-257
GetCodecInfo I-385
GetMaxCompressionSize I-420
GetCSequenceMaxCompressionSize I-405
GetCompressionTime I-401
CompressImage I-113
FCompressImage I-344
DecompressImage I-235
FDecompressImage I-355
CompressSequenceBegin I-119
CompressSequenceFrame I-124
DecompressSequenceBegin I-238
DecompressSequenceBeginS I-239
DecompressSequenceFrame I-242
DecompressSequenceFrameS I-243
DecompressSequenceFrameWhen I-245
CDSequenceFlush I-80
SetDSequenceMatrix III-1587
GetDSequenceMatrix I-410
SetDSequenceMatte III-1588
SetDSequenceMask III-1586
SetDSequenceTransferMode III-1591
SetDSequenceDataProc III-1584
SetDSequenceAccuracy III-1583
SetDSequenceSrcRect III-1589
SetDSequenceFlags III-1585
GetDSequenceImageBuffer I-409
GetDSequenceScreenBuffer I-410
SetCSequenceQuality III-1580
SetCSequencePrev III-1579
SetCSequenceFlushProc III-1575
SetCSequenceKeyFrameRate III-1577
GetCSequenceKeyFrameRate I-405
GetCSequencePrevBuffer I-406
CDSequenceBusy I-75
CDSequenceEnd I-77
CDSequenceEquivalentImageDescription I-78
GetCompressedImageSize I-396
GetSimilarity I-508
GetImageDescriptionCTable I-417
SetImageDescriptionCTable III-1593
GetImageDescriptionExtension I-418
AddImageDescriptionExtension I-25
RemoveImageDescriptionExtension III-1457
CountImageDescriptionExtensionType I-143
GetNextImageDescriptionExtensionType I-501
FindCodec I-358
CompressPicture I-116

FCompressPicture I-348
CompressPictureFile I-118
FCompressPictureFile I-352
GetPictureFileHeader I-504
DrawPictureFile I-310
DrawTrimmedPicture I-311
DrawTrimmedPictureFile I-313
MakeThumbnailFromPicture II-790
MakeThumbnailFromPictureFile II-791
MakeThumbnailFromPixMap II-792
TrimImage III-1956
ConvertImage I-131
GetCompressedPixMapInfo I-397
SetCompressedPixMapInfo III-1573
StdPix III-1912
TransformRgn III-1955
SFGetFilePreview III-1674
SFPGetFilePreview III-1676
StandardGetFilePreview III-1910
CustomGetFilePreview I-163
MakeFilePreview II-784
AddFilePreview I-24
AlignScreenRect I-46
AlignWindow I-47
DragAlignedWindow I-306
DragAlignedGrayRgn I-304
SetCSequenceDataRateParams III-1574
SetCSequenceFrameNumber III-1576
SetCSequencePreferredPacketSize III-1578
NewImageGWorld II-1066
GetCSequenceDataRateParams I-403
GetCSequenceFrameNumber I-404
GetBestDeviceRect I-382
SetSequenceProgressProc III-1639
GDHasScale I-381
GDGetScale I-380
GDSetScale I-382
ICMShieldSequenceCursor II-679
ICMDecompressComplete II-671
ICMDecompressCompleteS II-672
ICMSequenceLockBits II-675
ICMSequenceUnlockBits II-677
ICMGetPixelFormatInfo II-673
ICMSetPixelFormatInfo II-678
ICMSequenceGetChainMember II-674
SetDSequenceTimeCode III-1590
CDSequenceNewMemory I-84
CDSequenceDisposeMemory I-77
CDSequenceNewDataSource I-82

CDSequenceDisposeDataSource I-76
CDSequenceSetSourceData I-86
CDSequenceChangedSourceData I-76
CDSequenceSetSourceDataQueue I-87
CDSequenceGetDataSource I-81
PtInDSequenceData II-1147
HitTestDSequenceData I-667
GetGraphicsImporterForFile I-414
GetGraphicsImporterForDataRef I-412
GetGraphicsImporterForFileWithFlags I-416
GetGraphicsImporterForDataRefWithFlags I-413
QTGetFileNameExtension II-1177
ImageTranscodeSequenceBegin II-754
ImageTranscodeSequenceEnd II-755
ImageTranscodeFrame II-750
ImageTranscodeDisposeFrameData II-749
CDSequenceInvalidate I-82
CDSequenceSetTimeBase I-88
ImageFieldSequenceBegin II-746
ImageFieldSequenceExtractCombine II-747
ImageFieldSequenceEnd II-747
QTNewGWorld II-1204
QTNewGWorldFromPtr II-1206
QTUpdateGWorld II-1318
MakeImageDescriptionForPixMap II-787
MakeImageDescriptionForEffect II-785
QTGetPixelSize II-1179
QTGetPixMapPtrRowBytes II-1183
QTGetPixMapHandleRowBytes II-1181
QTSetPixMapPtrRowBytes II-1231
QTSetPixMapHandleRowBytes II-1228
QuadToQuadMatrix II-1447
GetMatrixType I-419
CopyMatrix I-139
EqualMatrix I-338
SetIdentityMatrix III-1593
TranslateMatrix III-1956
RotateMatrix III-1462
ScaleMatrix III-1527
SkewMatrix III-1827
TransformFixedPoints III-1951
TransformPoints III-1953
TransformFixedRect III-1952
TransformRect III-1954
InverseMatrix II-774
ConcatMatrix I-128
RectMatrix III-1451
MapMatrix II-794
CompAdd I-107

```

CompSub      I-127
CompNeg      I-111
CompShift    I-127
CompMul      I-109
CompDiv      I-108
CompFixMul   I-108
CompMulDiv   I-109
CompMulDivTrunc I-110
CompCompare  I-107
CompSquareRoot I-127
FixMulDiv    I-362
UnsignedFixMulDiv III-1981
FracSinCos   I-378
FixExp2      I-361
FixLog2      I-362
FixPow       I-363
GraphicsImportSetDataReference I-653
GraphicsImportGetDataReference I-626
GraphicsImportSetDataFile I-651
GraphicsImportGetDataFile I-621
GraphicsImportSetDataHandle I-652
GraphicsImportGetDataHandle I-623
GraphicsImportGetImageDescription I-638
GraphicsImportGetDataOffsetAndSize I-624
GraphicsImportReadData I-645
GraphicsImportSetClip I-650
GraphicsImportGetClip I-620
GraphicsImportSetSourceRect I-664
GraphicsImportGetSourceRect I-645
GraphicsImportGetNaturalBounds I-642
GraphicsImportDraw I-616
GraphicsImportSetGWorld I-660
GraphicsImportGetGWorld I-636
GraphicsImportSetMatrix I-661
GraphicsImportGetMatrix I-639
GraphicsImportSetBoundsRect I-649
GraphicsImportGetBoundsRect I-619
GraphicsImportSaveAsPicture I-647
GraphicsImportSetGraphicsMode I-659
GraphicsImportGetGraphicsMode I-635
GraphicsImportSetQuality I-663
GraphicsImportGetQuality I-643
GraphicsImportSaveAsQuickTimeImageFile I-648
GraphicsImportSetDataReferenceOffsetAndLimit I-654
GraphicsImportGetDataReferenceOffsetAndLimit I-627
GraphicsImportGetAliasedDataReference I-618
GraphicsImportValidate I-665
GraphicsImportGetMetaData I-640
GraphicsImportGetMIMETypeList I-641

```

GraphicsImportDoesDrawAllPixels I-613
GraphicsImportGetAsPicture I-618
GraphicsImportExportImageFile I-616
GraphicsImportGetExportImageTypeList I-632
GraphicsImportDoExportImageFileDialog I-614
GraphicsImportGetExportSettingsAsAtomContainer I-633
GraphicsImportSetExportSettingsFromAtomContainer I-657
GraphicsImportSetProgressProc I-662
GraphicsImportGetProgressProc I-643
GraphicsImportGetImageCount I-637
GraphicsImportSetImageIndex I-661
GraphicsImportGetImageIndex I-638
GraphicsImportGetDataOffsetAndSize64 I-625
GraphicsImportReadData64 I-646
GraphicsImportSetDataReferenceOffsetAndLimit64 I-656
GraphicsImportGetDataReferenceOffsetAndLimit64 I-628
GraphicsImportGetDefaultMatrix I-630
GraphicsImportGetDefaultClip I-629
GraphicsImportGetDefaultGraphicsMode I-630
GraphicsImportGetDefaultSourceRect I-631
GraphicsImportGetColorSyncProfile I-621
GraphicsImportSetDestRect I-657
GraphicsImportGetDestRect I-632
GraphicsImportSetFlags I-658
GraphicsImportGetFlags I-634
GraphicsExportDoExport I-554
GraphicsExportCanTranscode I-552
GraphicsExportDoTranscode I-555
GraphicsExportCanUseCompressor I-553
GraphicsExportDoUseCompressor I-556
GraphicsExportDoStandaloneExport I-554
GraphicsExportGetDefaultFileTypeAndCreator I-561
GraphicsExportGetDefaultFileNameExtension I-560
GraphicsExportGetMIMETypeList I-577
GraphicsExportRequestSettings I-588
GraphicsExportSetSettingsFromAtomContainer I-610
GraphicsExportGetSettingsAsAtomContainer I-583
GraphicsExportGetSettingsAsText I-584
GraphicsExportSetDontRecompress I-592
GraphicsExportGetDontRecompress I-562
GraphicsExportSetInterlaceStyle I-603
GraphicsExportGetInterlaceStyle I-575
GraphicsExportSetMetaData I-604
GraphicsExportGetMetaData I-576
GraphicsExportSetTargetDataSize I-611
GraphicsExportGetTargetDataSize I-585
GraphicsExportSetCompressionMethod I-589
GraphicsExportGetCompressionMethod I-559
GraphicsExportSetCompressionQuality I-590

GraphicsExportGetCompressionQuality I-559
 GraphicsExportSetResolution I-609
 GraphicsExportGetResolution I-583
 GraphicsExportSetDepth I-591
 GraphicsExportGetDepth I-562
 GraphicsExportSetColorSyncProfile I-589
 GraphicsExportGetColorSyncProfile I-558
 GraphicsExportSetProgressProc I-608
 GraphicsExportGetProgressProc I-582
 GraphicsExportSetInputDataReference I-593
 GraphicsExportGetInputDataReference I-563
 GraphicsExportSetInputFile I-594
 GraphicsExportGetInputFile I-565
 GraphicsExportSetInputHandle I-597
 GraphicsExportGetInputHandle I-568
 GraphicsExportSetInputPtr I-602
 GraphicsExportGetInputPtr I-574
 GraphicsExportSetInputGraphicsImporter I-595
 GraphicsExportGetInputGraphicsImporter I-566
 GraphicsExportSetInputPicture I-599
 GraphicsExportGetInputPicture I-572
 GraphicsExportSetInputGWorld I-596
 GraphicsExportGetInputGWorld I-567
 GraphicsExportSetInputPixmap I-600
 GraphicsExportGetInputPixmap I-573
 GraphicsExportSetInputOffsetAndLimit I-598
 GraphicsExportGetInputOffsetAndLimit I-572
 GraphicsExportMayExporterReadInputData I-585
 GraphicsExportGetInputDataSize I-564
 GraphicsExportReadInputData I-586
 GraphicsExportGetInputImageDescription I-570
 GraphicsExportGetInputImageDimensions I-571
 GraphicsExportGetInputImageDepth I-569
 GraphicsExportDrawInputImage I-557
 GraphicsExportSetOutputDataReference I-604
 GraphicsExportGetOutputDataReference I-578
 GraphicsExportSetOutputFile I-605
 GraphicsExportGetOutputFile I-578
 GraphicsExportSetOutputHandle I-606
 GraphicsExportGetOutputHandle I-580
 GraphicsExportSetOutputOffsetAndMaxSize I-608
 GraphicsExportGetOutputOffsetAndMaxSize I-581
 GraphicsExportSetOutputFileTypeAndCreator I-606
 GraphicsExportGetOutputFileTypeAndCreator I-579
 GraphicsExportWriteOutputData I-611
 GraphicsExportSetOutputMark I-607
 GraphicsExportGetOutputMark I-580
 GraphicsExportReadOutputData I-587
 ImageTranscoderBeginSequence II-751

ImageTranscoderConvert II-752
ImageTranscoderDisposeData II-753
ImageTranscoderEndSequence II-754

MediaHandlers.h

CallComponentExecuteWiredAction I-60
MediaInitialize II-877
MediaSetHandlerCapabilities II-894
MediaIdle II-874
MediaGetMediaInfo II-853
MediaPutMediaInfo II-884
MediaSetActive II-889
MediaSetRate II-903
MediaGGetStatus II-869
MediaTrackEdited II-914
MediaSetMediaTimeScale II-898
MediaSetMovieTimeScale II-899
MediaSetGWorld II-893
MediaSetDimensions II-891
MediaSetClip II-890
MediaSetMatrix II-897
MediaGetTrackOpaque II-865
MediaSetGraphicsMode II-892
MediaGetGraphicsMode II-852
MediaGSetVolume II-871
MediaSetSoundBalance II-904
MediaGetSoundBalance II-860
MediaGetNextBoundsChange II-855
MediaGetSrcRgn II-865
MediaPreroll II-883
MediaSampleDescriptionChanged II-887
MediaHasCharacteristic II-872
MediaGetOffscreenBufferSize II-858
MediaSetHints II-896
MediaGetName II-855
MediaForceUpdate II-847
MediaGetDrawingRgn II-849
MediaGSetActiveSegment II-870
MediaInvalidateRegion II-879
MediaGetNextStepTime II-856
MediaSetNonPrimarySourceData II-899
MediaChangedNonPrimarySource II-843
MediaTrackReferencesChanged II-915
MediaGetSampleDataPointer II-859
MediaReleaseSampleDataPointer II-886
MediaTrackPropertyAtomChanged II-915
MediaSetTrackInputMapReference II-908

MediaSetVideoParam II-910
 MediaGetVideoParam II-868
 MediaCompare II-844
 MediaGetClock II-849
 MediaSetSoundOutputComponent II-908
 MediaGetSoundOutputComponent II-864
 MediaSetSoundLocalizationData II-907
 MediaGetInvalidRegion II-853
 MediaSampleDescriptionB2N II-887
 MediaSampleDescriptionN2B II-888
 MediaQueueNonPrimarySourceData II-885
 MediaFlushNonPrimarySourceData II-847
 MediaGetURLLink II-866
 MediaMakeMediaTimeTable II-879
 MediaHitTestForTargetRefCon II-873
 MediaHitTestTargetRefCon II-873
 MediaGetActionsForQTEvent II-848
 MediaDisposeTargetRefCon II-845
 MediaTargetRefConsEqual II-912
 MediaSetActionsCallback II-888
 MediaPrePrerollBegin II-882
 MediaPrePrerollCancel II-883
 MediaEnterEmptyEdit II-846
 MediaCurrentMediaQueuedData II-844
 MediaGetEffectiveVolume II-851
 MediaResolveTargetRefCon II-886
 MediaGetSoundLevelMeteringEnabled II-863
 MediaSetSoundLevelMeteringEnabled II-906
 MediaGetSoundLevelMeterInfo II-863
 MediaGetEffectiveSoundBalance II-850
 MediaSetScreenLock II-904
 MediaSetDoMCActionCallback II-892
 MediaGetErrorString II-851
 MediaGetSoundEqualizerBands II-862
 MediaSetSoundEqualizerBands II-906
 MediaGetSoundEqualizerBandLevels II-862
 MediaDoIdleActions II-845
 MediaSetSoundBassAndTreble II-905
 MediaGetSoundBassAndTreble II-861
 MediaTimeBaseChanged II-913
 MediaMCIsPlayerEvent II-881
 MediaGetMediaLoadState II-854
 NewPrePrerollCompleteUPP II-1095
 DisposePrePrerollCompleteUPP I-280

Movies.h

CheckQuickTimeRegistration I-89

EnterMovies I-337
ExitMovies I-340
GetMoviesError I-492
ClearMoviesStickyError I-90
GetMoviesStickyError I-494
SetMoviesErrorProc III-1633
MoviesTask II-973
PrerollMovie II-1142
PrePrerollMovie II-1141
AbortPrePrerollMovie I-21
LoadMovieIntoRam II-781
LoadTrackIntoRam II-782
LoadMediaIntoRam II-779
SetMovieActive III-1609
GetMovieActive I-456
StartMovie III-1911
StopMovie III-1914
GoToBeginningOfMovie I-550
GoToEndOfMovie I-551
IsMovieDone II-774
GetMoviePreviewMode I-487
SetMoviePreviewMode III-1628
ShowMoviePoster III-1827
PlayMoviePreview II-1140
GetMovieTimeBase I-496
SetMovieMasterTimeBase III-1621
SetMovieMasterClock III-1620
GetMovieGWorld I-471
SetMovieGWorld III-1619
SetMovieDrawingCompleteProc III-1617
GetMovieNaturalBoundsRect I-479
GetNextTrackForCompositing I-502
GetPrevTrackForCompositing I-506
SetTrackGWorld III-1659
GetMoviePict I-483
GetTrackPict I-540
GetMoviePosterPict I-484
UpdateMovie III-1981
InvalidateMovieRegion II-771
GetMovieBox I-460
SetMovieBox III-1611
GetMovieDisplayClipRgn I-469
SetMovieDisplayClipRgn III-1616
GetMovieClipRgn I-462
SetMovieClipRgn III-1612
GetTrackClipRgn I-524
SetTrackClipRgn III-1656
GetMovieDisplayBoundsRgn I-468
GetTrackDisplayBoundsRgn I-528

GetMovieBoundsRgn I-459
 GetTrackMovieBoundsRgn I-537
 GetTrackBoundsRgn I-523
 GetTrackMatte I-534
 SetTrackMatte III-1663
 DisposeMatte I-267
 NewMovie II-1069
 PutMovieIntoHandle II-1151
 PutMovieIntoDataFork II-1150
 PutMovieIntoDataFork64 II-1150
 DisposeMovie I-268
 GetMovieCreationTime I-465
 GetMovieModificationTime I-479
 GetMovieTimeScale I-496
 SetMovieTimeScale III-1635
 GetMovieDuration I-470
 GetMovieRate I-490
 SetMovieRate III-1631
 GetMoviePreferredRate I-485
 SetMoviePreferredRate III-1626
 GetMoviePreferredVolume I-486
 SetMoviePreferredVolume III-1627
 GetMovieVolume I-500
 SetMovieVolume III-1637
 GetMovieMatrix I-478
 SetMovieMatrix III-1622
 GetMoviePreviewTime I-487
 SetMoviePreviewTime III-1629
 GetMoviePosterTime I-485
 SetMoviePosterTime III-1625
 GetMovieSelection I-491
 SetMovieSelection III-1632
 SetMovieActiveSegment III-1609
 GetMovieActiveSegment I-457
 GetMovieTime I-495
 SetMovieTime III-1634
 SetMovieTimeValue III-1635
 GetMovieUserData I-499
 GetMovieTrackCount I-498
 GetMovieTrack I-497
 GetMovieIndTrack I-473
 GetMovieIndTrackType I-475
 GetTrackID I-531
 GetTrackMovie I-536
 NewMovieTrack II-1092
 DisposeMovieTrack I-278
 GetTrackCreationTime I-524
 GetTrackModificationTime I-536
 GetTrackEnabled I-531

SetTrackEnabled III-1658
GetTrackUsage I-545
SetTrackUsage III-1668
GetTrackDuration I-529
GetTrackOffset I-539
SetTrackOffset III-1664
GetTrackLayer I-532
SetTrackLayer III-1660
GetTrackAlternate I-522
SetTrackAlternate III-1656
SetAutoTrackAlternatesEnabled III-1570
SelectMovieAlternates III-1569
GetTrackVolume I-547
SetTrackVolume III-1669
GetTrackMatrix I-533
SetTrackMatrix III-1662
GetTrackDimensions I-527
SetTrackDimensions III-1657
GetTrackUserData I-546
GetTrackDisplayMatrix I-528
GetTrackSoundLocalizationSettings I-543
SetTrackSoundLocalizationSettings III-1666
NewTrackMedia II-1120
DisposeTrackMedia I-301
GetTrackMedia I-535
GetMediaTrack I-455
GetMediaCreationTime I-424
GetMediaModificationTime I-437
GetMediaTimeScale I-455
SetMediaTimeScale III-1608
GetMediaDuration I-430
GetMediaLanguage I-436
SetMediaLanguage III-1600
GetMediaQuality I-441
SetMediaQuality III-1605
GetMediaHandlerDescription I-432
GetMediaUserData I-456
GetMediaInputMap I-435
SetMediaInputMap III-1599
GetMediaHandler I-432
SetMediaHandler III-1598
BeginMediaEdits I-56
EndMediaEdits I-335
SetMediaDefaultDataRefIndex III-1597
GetMediaDataHandlerDescription I-426
GetMediaDataHandler I-425
SetMediaDataHandler III-1594
GetDataHandler I-407
OpenADataHandler II-1127

GetMediaSampleDescriptionCount I-447
 GetMediaSampleDescription I-445
 SetMediaSampleDescription III-1606
 GetMediaSampleCount I-445
 GetMediaSyncSampleCount I-454
 SampleNumToMediaTime III-1526
 MediaTimeToSampleNum II-913
 AddMediaSample I-27
 AddMediaSampleReference I-31
 AddMediaSampleReferences I-33
 AddMediaSampleReferences64 I-34
 GetMediaSample I-443
 GetMediaSampleReference I-447
 GetMediaSampleReferences I-449
 GetMediaSampleReferences64 I-451
 SetMediaPreferredChunkSize III-1603
 GetMediaPreferredChunkSize I-440
 SetMediaShadowSync III-1607
 GetMediaShadowSync I-453
 InsertMediaIntoTrack II-761
 InsertTrackSegment II-764
 InsertMovieSegment II-762
 InsertEmptyTrackSegment II-760
 InsertEmptyMovieSegment II-759
 DeleteTrackSegment I-252
 DeleteMovieSegment I-250
 ScaleTrackSegment III-1529
 ScaleMovieSegment III-1528
 CutMovieSelection I-166
 CopyMovieSelection I-139
 PasteMovieSelection II-1139
 AddMovieSelection I-38
 ClearMovieSelection I-90
 PasteHandleIntoMovie II-1137
 PutMovieIntoTypedHandle II-1152
 IsScrapMovie II-775
 CopyTrackSettings I-141
 CopyMovieSettings I-140
 AddEmptyTrackToMovie I-23
 NewMovieEditState II-1073
 UseMovieEditState III-1984
 DisposeMovieEditState I-273
 NewTrackEditState II-1119
 UseTrackEditState III-1984
 DisposeTrackEditState I-300
 AddTrackReference I-42
 DeleteTrackReference I-252
 SetTrackReference III-1665
 GetTrackReference I-541

GetNextTrackReferenceType I-503
GetTrackReferenceCount I-542
ConvertFileToMovieFile I-129
ConvertMovieToFile I-134
GetMovieImporterForDataRef I-472
TrackTimeToMediaTime III-1951
GetTrackEditRate I-530
GetMovieDataSize I-465
GetMovieDataSize64 I-466
GetTrackDataSize I-525
GetTrackDataSize64 I-526
GetMediaDataSize I-429
GetMediaDataSize64 I-430
PtInMovie II-1148
PtInTrack II-1149
SetMovieLanguage III-1620
GetUserData I-547
AddUserData I-44
RemoveUserData III-1459
CountUserDataTypes I-144
GetNextUserDataTypes I-503
GetUserDataItem I-548
SetUserDataItem III-1673
AddUserDataText I-45
GetUserDataText I-549
RemoveUserDataText III-1460
NewUserData II-1124
DisposeUserData I-303
NewUserDataFromHandle II-1124
PutUserDataIntoHandle II-1154
SetMoviePropertyAtom III-1631
GetMoviePropertyAtom I-489
GetMediaNextInterestingTime I-437
GetTrackNextInterestingTime I-537
GetMovieNextInterestingTime I-480
CreateMovieFile I-145
OpenMovieFile II-1133
CloseMovieFile I-102
DeleteMovieFile I-249
NewMovieFromFile II-1080
NewMovieFromHandle II-1084
NewMovieFromDataFork II-1075
NewMovieFromDataFork64 II-1077
NewMovieFromUserProc II-1087
NewMovieFromDataRef II-1078
AddMovieResource I-36
UpdateMovieResource III-1983
RemoveMovieResource III-1458
HasMovieChanged I-666

ClearMovieChanged I-89
 SetMovieDefaultDataRef III-1616
 GetMovieDefaultDataRef I-467
 SetMovieAnchorDataRef III-1610
 GetMovieAnchorDataRef I-458
 SetMovieColorTable III-1613
 GetMovieColorTable I-463
 FlattenMovie I-372
 FlattenMovieData I-374
 SetMovieProgressProc III-1630
 GetMovieProgressProc I-488
 CreateShortcutMovieFile I-149
 MovieSearchText II-972
 GetPosterBox I-505
 SetPosterBox III-1638
 GetMovieSegmentDisplayBoundsRgn I-490
 GetTrackSegmentDisplayBoundsRgn I-542
 SetMovieCoverProcs III-1614
 GetMovieCoverProcs I-464
 GetTrackStatus I-544
 GetMovieStatus I-494
 GetMovieLoadState I-477
 NewMovieController II-1071
 DisposeMovieController I-270
 ShowMovieInformation III-1826
 PutMovieOnScrap II-1153
 NewMovieFromScrap II-1085
 GetMediaDataRef I-427
 SetMediaDataRef III-1595
 SetMediaDataRefAttributes III-1596
 AddMediaDataRef I-26
 GetMediaDataRefCount I-428
 QTNewAlias II-1202
 SetMoviePlayHints III-1623
 SetMediaPlayHints III-1601
 GetMediaPlayHints I-439
 SetTrackLoadSettings III-1661
 GetTrackLoadSettings I-532
 BeginFullScreen I-52
 EndFullScreen I-314
 AddMovieExecuteWiredActionsProc I-36
 RemoveMovieExecuteWiredActionsProc III-1457
 MovieExecuteWiredActions II-918
 NewSpriteWorld II-1114
 DisposeSpriteWorld I-297
 SetSpriteWorldClip III-1644
 SetSpriteWorldMatrix III-1647
 SetSpriteWorldGraphicsMode III-1646
 SpriteWorldIdle III-1908

InvalidSpriteWorld II-773
SpriteWorldHitTest III-1907
SpriteHitTest III-1885
DisposeAllSprites I-255
SetSpriteWorldFlags III-1645
NewSprite II-1113
DisposeSprite I-296
InvalidSprite II-772
SetSpriteProperty III-1641
GetSpriteProperty I-512
QTNewAtomContainer II-1203
QTDisposeAtomContainer II-1164
QTGetNextChildType II-1178
QTCountChildrenOfType II-1160
QTFindChildByIndex II-1167
QTFindChildByID II-1166
QTNextChildAnyType II-1209
QTSetAtomData II-1223
QTCopyAtomDataToHandle II-1158
QTCopyAtomDataToPtr II-1159
QTGetAtomTypeAndID II-1172
QTCopyAtom II-1158
QTLockContainer II-1187
QTGetAtomDataPtr II-1171
QTUnlockContainer II-1316
QTInsertChild II-1183
QTInsertChildren II-1185
QTRemoveAtom II-1215
QTRemoveChildren II-1216
QTReplaceAtom II-1217
QTSwapAtoms II-1315
QTSetAtomID II-1225
QTGetAtomParent II-1172
SetMediaPropertyAtom III-1604
GetMediaPropertyAtom I-440
QTNewTween II-1209
QTDisposeTween II-1164
QTDoTween II-1165
AddSoundDescriptionExtension I-40
GetSoundDescriptionExtension I-509
RemoveSoundDescriptionExtension III-1459
GetQuickTimePreference I-507
SetQuickTimePreference III-1639
QTGetEffectsList II-1174
QTCreateStandardParameterDialog II-1161
QTIsStandardParameterDialogEvent II-1186
QTDismissStandardParameterDialog II-1163
QTStandardParameterDialogDoAction II-1313
QTGetEffectSpeed II-1176

QTGetAccessKeys II-1168
 QTRegisterAccessKey II-1214
 QTUnregisterAccessKey II-1317
 MakeTrackTimeTable II-793
 MakeMediaTimeTable II-788
 GetMaxLoadedTimeInMovie I-423
 QTMovieNeedsTimeTable II-1202
 QTGetDataRefMaxFileOffset II-1173
 QTBandwidthRequest II-1156
 QTBandwidthRequestForTimeBase II-1157
 QTBandwidthRelease II-1155
 QTScheduledBandwidthRequest II-1219
 QTScheduledBandwidthRelease II-1219
 ITextAddString II-776
 ITextRemoveString II-778
 ITextGetString II-777
 QTTextToNativeText II-1315
 QTParseTextHREF II-1210
 VideoMediaResetStatistics III-2059
 VideoMediaGetStatistics III-2059
 VideoMediaGetStallCount III-2058
 VideoMediaSetCodecParameter III-2060
 VideoMediaGetCodecParameter III-2057
 TextMediaSetTextProc III-1947
 TextMediaAddTESample III-1933
 TextMediaAddHiliteSample III-1932
 TextMediaDrawRaw III-1940
 TextMediaSetTextProperty III-1948
 TextMediaRawSetup III-1946
 TextMediaRawIdle III-1945
 TextMediaFindNextText III-1941
 TextMediaHiliteTextSample III-1944
 TextMediaSetTextSampleData III-1950
 SpriteMediaSetProperty III-1903
 SpriteMediaGetProperty III-1893
 SpriteMediaHitTestSprites III-1900
 SpriteMediaCountSprites III-1887
 SpriteMediaCountImages III-1886
 SpriteMediaGetIndImageDescription III-1891
 SpriteMediaGetDisplayedSampleNumber III-1889
 SpriteMediaGetSpriteName III-1895
 SpriteMediaGetImageName III-1890
 SpriteMediaSetSpriteProperty III-1904
 SpriteMediaGetSpriteProperty III-1896
 SpriteMediaHitTestAllSprites III-1897
 SpriteMediaHitTestOneSprite III-1898
 SpriteMediaSpriteIndexToID III-1906
 SpriteMediaSpriteIDToIndex III-1906
 SpriteMediaGetSpriteActionsForQTEvent III-1894

SpriteMediaSetActionVariable III-1901
SpriteMediaGetActionVariable III-1888
SpriteMediaGetIndImageProperty III-1891
SpriteMediaNewSprite III-1901
SpriteMediaDisposeSprite III-1887
SpriteMediaSetActionVariableToString III-1902
SpriteMediaGetActionVariableAsString III-1889
FlashMediaSetPan I-370
FlashMediaSetZoom I-370
FlashMediaSetZoomRect I-371
FlashMediaGetRefConBounds I-366
FlashMediaGetRefConID I-367
FlashMediaIDToRefCon I-368
FlashMediaGetDisplayedFrameNumber I-365
FlashMediaFrameNumberToMovieTime I-364
FlashMediaFrameLabelToMovieTime I-364
MovieMediaGetChildDoMCActionCallback II-966
MovieMediaGetDoMCActionCallback II-970
MovieMediaGetCurrentMovieProperty II-968
MovieMediaGetCurrentTrackProperty II-969
MovieMediaGetChildMovieDataReference II-967
MovieMediaSetChildMovieDataReference II-971
MovieMediaLoadChildMovieFromDataReference II-970
Media3DGetNamedObjectList II-840
Media3DGetRendererList II-840
Media3DGetCurrentGroup II-839
Media3DTranslateNamedObjectTo II-843
Media3DScaleNamedObjectTo II-841
Media3DRotateNamedObjectTo II-841
Media3DSetCameraData II-842
Media3DGetCameraData II-839
Media3DSetCameraAngleAspect II-842
Media3DGetCameraAngleAspect II-838
Media3DSetCameraRange II-842
Media3DGetCameraRange II-839
Media3DGetViewObject II-841
MCSetMovie II-835
MCGetIndMovie II-812
MCRemoveAllMovies II-827
MCRemoveAMovie II-827
MCRemoveMovie II-828
MCIsPlayerEvent II-819
MCSetActionFilter II-829
MCDoAction II-801
MCSetControllerAttached II-831
MCIsControllerAttached II-818
MCSetControllerPort II-833
MCGetControllerPort II-810
MCSetVisible II-837

MCGetVisible II-815
 MCGetControllerBoundsRect II-806
 MCSetControllerBoundsRect II-832
 MCGetControllerBoundsRgn II-807
 MCGetWindowRgn II-815
 MCMovieChanged II-822
 MCSetDuration II-834
 MCGetCurrentTime II-810
 MCNewAttachedController II-823
 MCDraw II-803
 MCActivate II-795
 MCIdle II-816
 MCKey II-821
 MCClick II-799
 MCEnableEditing II-805
 MCIsEditingEnabled II-818
 MCCopy II-800
 MCCut II-800
 MCPaste II-824
 MCClear II-798
 MCUndo II-838
 MCPositionController II-824
 MCGetControllerInfo II-808
 MCSetClip II-830
 MCGetClip II-805
 MCDrawBadge II-804
 MCSetUpEditMenu II-836
 MCGetMenuString II-814
 MCSetActionFilterWithRefCon II-829
 MCPtInController II-826
 MCInvalidate II-817
 MCAdjustCursor II-796
 MCGetInterfaceElement II-812
 MCGetDoActionsProc II-811
 NewTimeBase II-1119
 DisposeTimeBase I-299
 GetTimeBaseTime I-521
 SetTimeBaseTime III-1653
 SetTimeBaseValue III-1654
 GetTimeBaseRate I-518
 SetTimeBaseRate III-1651
 GetTimeBaseStartTime I-518
 SetTimeBaseStartTime III-1652
 GetTimeBaseStopTime I-520
 SetTimeBaseStopTime III-1652
 GetTimeBaseFlags I-515
 SetTimeBaseFlags III-1648
 SetTimeBaseMasterTimeBase III-1650
 GetTimeBaseMasterTimeBase I-517

SetTimeBaseMasterClock III-1649
GetTimeBaseMasterClock I-516
ConvertTime I-137
ConvertTimeScale I-138
AddTime I-41
SubtractTime III-1915
GetTimeBaseStatus I-519
SetTimeBaseZero III-1655
GetTimeBaseEffectiveRate I-514
NewCallBack II-1053
DisposeCallBack I-256
GetCallBackType I-384
GetCallBackTimeBase I-384
CallMeWhen I-67
CancelCallBack I-70
AddCallBackToTimeBase I-21
RemoveCallBackFromTimeBase III-1456
GetFirstCallBack I-411
GetNextCallBack I-501
ExecuteCallBack I-339
MusicMediaGetIndexedTunePlayer II-1004
NewMovieRgnCoverUPP II-1091
NewMovieProgressUPP II-1090
NewMovieDrawingCompleteUPP II-1072
NewTrackTransferUPP II-1122
NewGetMovieUPP II-1060
NewMoviePreviewCallOutUPP II-1090
NewTextMediaUPP II-1118
NewActionsUPP II-1053
NewDoMCActionUPP II-1058
NewMovieExecuteWiredActionsUPP II-1073
NewMoviePrePrerollCompleteUPP II-1089
NewMoviesErrorUPP II-1091
NewQTCallBackUPP II-1097
NewQTSyncTaskUPP II-1099
NewTweenDataUPP II-1123
NewQTBandwidthNotificationUPP II-1096
NewMCActionFilterUPP II-1068
NewMCActionFilterWithRefConUPP II-1069
DisposeMovieRgnCoverUPP I-277
DisposeMovieProgressUPP I-276
DisposeMovieDrawingCompleteUPP I-273
DisposeTrackTransferUPP I-301
DisposeGetMovieUPP I-261
DisposeMoviePreviewCallOutUPP I-276
DisposeTextMediaUPP I-299
DisposeActionsUPP I-255
DisposeDoMCActionUPP I-259
DisposeMovieExecuteWiredActionsUPP I-274

DisposeMoviePrePrerollCompleteUPP I-275
 DisposeMoviesErrorUPP I-277
 DisposeQTCallBackUPP I-282
 DisposeQTSyncTaskUPP I-284
 DisposeTweenDataUPP I-302
 DisposeQTBandwidthNotificationUPP I-282
 DisposeMCActionFilterUPP I-268
 DisposeMCActionFilterWithRefConUPP I-268

MoviesFormat.h

No functions are declared in MoviesFormat.h.

QTML.h

QTMLYieldCPU II-1200
 QTMLYieldCPUTime II-1201
 InitializeQTML II-756
 TerminateQTML III-1926
 CreatePortAssociation I-148
 DestroyPortAssociation I-254
 QTMLGrabMutex II-1195
 QTMLTryGrabMutex II-1199
 QTMLReturnMutex II-1197
 QTMLCreateMutex II-1191
 QTMLDestroyMutex II-1192
 QTMLCreateSyncVar II-1192
 QTMLDestroySyncVar II-1193
 QTMLTestAndSetSyncVar II-1198
 QTMLWaitAndSetSyncVar II-1200
 QTMLResetSyncVar II-1197
 InitializeQHdr II-756
 TerminateQHdr III-1926
 QTMLAcquireWindowList II-1191
 QTMLReleaseWindowList II-1196
 QTMLRegisterInterruptSafeThread II-1196
 QTMLUnregisterInterruptSafeThread II-1199
 NativeEventToMacEvent II-1049
 WinEventToMacEvent III-2061
 IsTaskBarVisible II-776
 ShowHideTaskBar III-1825
 QTGetDDObject II-1174
 QTSetDDObject II-1225
 QTSetDDPrimarySurface II-1226
 QTMLGetVolumeRootPath II-1194
 QTMLSetWindowWndProc II-1198

QTMLGetWindowWndProc II-1195
 QTMLGetCanonicalPathName II-1193
 FSSpecToNativePathName I-379
 NativePathNameToFSSpec II-1050

QTSMovie.h

QTSMediaSetInfo II-1248
 QTSMediaGetInfo II-1245
 QTSMediaSetIndStreamInfo II-1246
 QTSMediaGetIndStreamInfo II-1244

QTStreamingComponents.h

QTSFindReassemblerForPayloadID II-1234
 QTSFindReassemblerForPayloadName II-1235
 RTPRssmInitialize III-1514
 RTPRssmHandleNewPacket III-1512
 RTPRssmComputeChunkSize III-1502
 RTPRssmAdjustPacketParams III-1500
 RTPRssmCopyDataToChunk III-1503
 RTPRssmSendPacketList III-1518
 RTPRssmGetTimeScaleFromPacket III-1511
 RTPRssmSetInfo III-1521
 RTPRssmGetInfo III-1507
 RTPRssmHasCharacteristic III-1512
 RTPRssmReset III-1516
 RTPRssmSetCapabilities III-1520
 RTPRssmGetCapabilities III-1505
 RTPRssmSetPayloadHeaderLength III-1523
 RTPRssmGetPayloadHeaderLength III-1509
 RTPRssmSetTimeScale III-1525
 RTPRssmGetTimeScale III-1510
 RTPRssmNewStreamHandler III-1514
 RTPRssmSetStreamHandler III-1524
 RTPRssmGetStreamHandler III-1510
 RTPRssmSendStreamHandlerChanged III-1520
 RTPRssmSetSampleDescription III-1524
 RTPRssmGetChunkAndIncrRefCount III-1506
 RTPRssmSendChunkAndDecrRefCount III-1517
 RTPRssmSendLostChunk III-1517
 RTPRssmSendStreamBufferRange III-1519
 RTPRssmClearCachedPackets III-1501
 RTPRssmFillPacketListParams III-1504
 RTPRssmReleasePacketList III-1515
 RTPRssmIncrChunkRefCount III-1513

RTPRssmDecrChunkRefCount III-1504
 QTSFindMediaPacketizer II-1231
 QTSFindMediaPacketizerForTrack II-1233
 QTSFindMediaPacketizerForPayloadID II-1232
 QTSFindMediaPacketizerForPayloadName II-1233
 RTPMPInitialize III-1473
 RTPMPPreflightMedia III-1474
 RTPMPIdle III-1472
 RTPMPSetSampleData III-1480
 RTPMPFlush III-1464
 RTPMPReset III-1474
 RTPMPSetInfo III-1475
 RTPMPGetInfo III-1464
 RTPMPSetTimeScale III-1482
 RTPMPGetTimeScale III-1470
 RTPMPSetTimeBase III-1481
 RTPMPGetTimeBase III-1470
 RTPMPHasCharacteristic III-1471
 RTPMPSetPacketBuilder III-1479
 RTPMPGetPacketBuilder III-1468
 RTPMPSetMediaType III-1478
 RTPMPGetMediaType III-1467
 RTPMPSetMaxPacketSize III-1477
 RTPMPGetMaxPacketSize III-1466
 RTPMPSetMaxPacketDuration III-1477
 RTPMPGetMaxPacketDuration III-1466
 RTPMPDoUserDialog III-1463
 RTPMPSetSettingsFromAtomContainerAtAtom III-1481
 RTPMPGetSettingsIntoAtomContainerAtAtom III-1469
 RTPMPGetSettingsAsText III-1468
 RTPPBBeginPacketGroup III-1490
 RTPPBEndPacketGroup III-1492
 RTPPBBeginPacket III-1489
 RTPPBEndPacket III-1491
 RTPPBAddPacketLiteralData III-1483
 RTPPBAddPacketSampleData III-1485
 RTPPBAddPacketRepeatedData III-1484
 RTPPBReleaseRepeatedData III-1497
 RTPPBSetPacketSequenceNumber III-1499
 RTPPBGetPacketSequenceNumber III-1494
 RTPPBSetCallback III-1497
 RTPPBGetCallback III-1493
 RTPPBSetInfo III-1498
 RTPPBGetInfo III-1493
 NewRTPMPDataReleaseUPP II-1102
 NewRTPPBCallbackUPP II-1103
 DisposeRTPMPDataReleaseUPP I-287
 DisposeRTPPBCallbackUPP I-287

QuickTimeComponents.h

ClockGetTime I-95
ClockNewCallBack I-96
ClockDisposeCallBack I-94
ClockCallMeWhen I-91
ClockCancelCallBack I-93
ClockRateChanged I-97
ClockTimeChanged I-100
ClockSetTimeBase I-98
ClockStartStopChanged I-99
ClockGetRate I-95
SCGetCompressionExtended III-1544
SCPositionRect III-1554
SCPositionDialog III-1553
SCSetTestImagePictHandle III-1566
SCSetTestImagePictFile III-1564
SCSetTestImagePixMap III-1568
SCGetBestDeviceRect III-1542
SCRequestImageSettings III-1555
SCCompressImage III-1530
SCCompressPicture III-1531
SCCompressPictureFile III-1532
SCRequestSequenceSettings III-1556
SCCompressSequenceBegin III-1533
SCCompressSequenceFrame III-1534
SCCompressSequenceEnd III-1534
SCDefaultPictHandleSettings III-1540
SCDefaultPictFileSettings III-1540
SCDefaultPixMapSettings III-1541
SCGetInfo III-1545
SCSetInfo III-1558
SCNewGWorld III-1552
SCSetCompressFlags III-1557
SCGetCompressFlags III-1543
SCGetSettingsAsText III-1551
SCGetSettingsAsAtomContainer III-1551
SCSetSettingsFromAtomContainer III-1564
TweenersInitialize III-1977
TweenersDoTween III-1976
TweenersReset III-1978
TCGetCurrentTimeCode III-1917
TCGetTimeCodeAtTime III-1920
TCTimeCodeToString III-1925
TCTimeCodeToFrameNumber III-1924
TCFrameNumberToTimeCode III-1916
TCGetSourceRef III-1919
TCSetSourceRef III-1922
TCSetTimeCodeFlags III-1923

TCGetTimeCodeFlags III-1921
 TCSetDisplayOptions III-1922
 TCGetDisplayOptions III-1918
 NewSCModalFilterUPP II-1103
 NewSCModalHookUPP II-1104
 NewMovieExportGetDataUPP II-1074
 NewMovieExportGetPropertyUPP II-1074
 DisposeSCModalFilterUPP I-288
 DisposeSCModalHookUPP I-288
 DisposeMovieExportGetDataUPP I-274
 DisposeMovieExportGetPropertyUPP I-275
 MovieImportHandle II-950
 MovieImportFile II-942
 MovieImportSetSampleDuration II-962
 MovieImportSetSampleDescription II-961
 MovieImportSetMediaFile II-958
 MovieImportSetDimensions II-955
 MovieImportSetChunkSize II-954
 MovieImportSetProgressProc II-961
 MovieImportSetAuxiliaryData II-954
 MovieImportSetFromScrap II-957
 MovieImportDoUserDialog II-940
 MovieImportSetDuration II-957
 MovieImportGetAuxiliaryDataType II-944
 MovieImportValidate II-964
 MovieImportGetFileType II-946
 MovieImportDataRef II-938
 MovieImportGetSampleDescription II-949
 MovieImportGetMIMETypeList II-948
 MovieImportSetOffsetAndLimit II-959
 MovieImportGetSettingsAsAtomContainer II-949
 MovieImportSetSettingsFromAtomContainer II-963
 MovieImportSetOffsetAndLimit64 II-960
 MovieImportIdle II-953
 MovieImportValidateDataRef II-965
 MovieImportGetLoadState II-947
 MovieImportGetMaxLoadedTime II-947
 MovieExportToHandle II-936
 MovieExportToFile II-934
 MovieExportGetAuxiliaryData II-923
 MovieExportSetProgressProc II-930
 MovieExportSetSampleDescription II-931
 MovieExportDoUserDialog II-921
 MovieExportGetCreatorType II-924
 MovieExportToDataRef II-933
 MovieExportFromProceduresToDataRef II-922
 MovieExportAddDataSource II-919
 MovieExportValidate II-937
 MovieExportGetSettingsAsAtomContainer II-925

MovieExportSetSettingsFromAtomContainer II-932
MovieExportGetFileNameExtension II-925
MovieExportGetShortFileTypeString II-926
MovieExportGetSourceMediaType II-927
MovieExportSetGetMoviePropertyProc II-929
TextExportGetDisplayData III-1928
TextExportGetTimeFraction III-1929
TextExportSetTimeFraction III-1931
TextExportGetSettings III-1928
TextExportSetSettings III-1930
MIDIImportGetSettings II-917
MIDIImportSetSettings II-917
MovieExportNewGetDataAndPropertiesProcs II-927
MovieExportDisposeGetDataAndPropertiesProcs II-920
GraphicsImageImportSetSequenceEnabled I-612
GraphicsImageImportGetSequenceEnabled I-612
PreviewShowData II-1146
PreviewMakePreview II-1145
PreviewMakePreviewReference II-1145
PreviewEvent II-1144
DataCodecDecompress I-170
DataCodecGetCompressBufferSize I-172
DataCodecCompress I-167
DataCodecBeginInterruptSafe I-166
DataCodecEndInterruptSafe I-172
DataCodecDecompressPartial I-171
DataCodecCompressPartial I-169
DataHGetData I-186
DataHPutData I-216
DataHFlushData I-184
DataHOpenForWrite I-210
DataHCloseForWrite I-177
DataHOpenForRead I-209
DataHCloseForRead I-176
DataHSetDataRef I-224
DataHGetDataRef I-189
DataHCompareDataRef I-178
DataHTask I-231
DataHScheduleData I-220
DataHFinishData I-182
DataHFlushCache I-184
DataHResolveDataRef I-218
DataHGetFileSize I-194
DataHCanUseDataRef I-175
DataHGetVolumeList I-207
DataHWrite I-232
DataHPreextend I-214
DataHSetFileSize I-227
DataHGetFreeSpace I-198

DataHCreateFile I-180
 DataHGetPreferredBlockSize I-204
 DataHGetDeviceIndex I-193
 DataHIsStreamingDataHandler I-208
 DataHGetDataInBuffer I-188
 DataHGetScheduleAheadTime I-206
 DataHSetCacheSizeLimit I-224
 DataHGetCacheSizeLimit I-185
 DataHGetMovie I-202
 DataHAddMovie I-173
 DataHUpdateMovie I-231
 DataHDoesBuffer I-182
 DataHGetFileName I-193
 DataHGetAvailableFileSize I-185
 DataHGetMacOSFileType I-200
 DataHGetMIMEType I-201
 DataHSetDataRefWithAnchor I-226
 DataHGetDataRefWithAnchor I-192
 DataHSetMacOSFileType I-229
 DataHSetTimeBase I-229
 DataHGetInfoFlags I-200
 DataHScheduleData64 I-222
 DataHWrite64 I-234
 DataHGetFileSize64 I-195
 DataHPreextend64 I-215
 DataHSetFileSize64 I-228
 DataHGetFreeSpace64 I-199
 DataHAppend64 I-174
 DataHReadAsync I-217
 DataHPollRead I-213
 DataHGetDataAvailability I-187
 DataHGetFileSizeAsync I-196
 DataHGetDataRefAsType I-190
 DataHSetDataRefExtension I-225
 DataHGetDataRefExtension I-191
 DataHGetMovieWithFlags I-203
 DataHPlaybackHints I-211
 DataHPlaybackHints64 I-212
 VDGetMaxSrcRect III-2012
 VDGetActiveSrcRect III-1989
 VDSetsDigitizerRect III-2038
 VDGetDigitizerRect III-1999
 VDGetVBlankRect III-2021
 VDGetMaskPixMap III-2011
 VDGetPlayThruDestination III-2014
 VDUseThisCLUT III-2057
 VDSetInputGammaValue III-2044
 VDGetInputGammaValue III-2006
 VDSetBrightness III-2032

VDGetBrightness III-1991
VDSetContrast III-2036
VDSetHue III-2041
VDSetSharpness III-2053
VDSetSaturation III-2053
VDGetContrast III-1995
VDGetHue III-2002
VDGetSharpness III-2018
VDGetSaturation III-2017
VDGrabOneFrame III-2024
VDGetMaxAuxBuffer III-2011
VDGetDigitizerInfo III-1998
VDGetCurrentFlags III-1996
VDSetKeyColor III-2046
VDGetKeyColor III-2008
VDAAddKeyColor III-1985
VDGetNextKeyColor III-2013
VDSetKeyColorRange III-2046
VDGetKeyColorRange III-2009
VDSetDigitizerUserInterrupt III-2039
VDSetInputColorSpaceMode III-2043
VDGetInputColorSpaceMode III-2004
VDSetClipState III-2033
VDGetClipState III-1991
VDSetClipRgn III-2032
VDClearClipRgn III-1986
VDGetCLUTInUse III-1992
VDSetPLLFilterType III-2051
VDGetPLLFilterType III-2015
VDGetMaskandValue III-2009
VDSetMasterBlendLevel III-2047
VDSetPlayThruDestination III-2048
VDSetPlayThruOnOff III-2050
VDSetFieldPreference III-2040
VDGetFieldPreference III-2001
VDPreflightDestination III-2026
VDPreflightGlobalRect III-2028
VDSetPlayThruGlobalRect III-2049
VDSetInputGammaRecord III-2044
VDGetInputGammaRecord III-2006
VDSetBlackLevelValue III-2031
VDGetBlackLevelValue III-1990
VDSetWhiteLevelValue III-2055
VDGetWhiteLevelValue III-2024
VDGetVideoDefaults III-2022
VDGetNumberOfInputs III-2013
VDGetInputFormat III-2005
VDSetInput III-2042
VDGetInput III-2003

VDSetInputStandard III-2045
 VDSetupBuffers III-2055
 VDGrabOneFrameAsync III-2025
 VDDone III-1988
 VDSetCompression III-2034
 VDCompressOneFrameAsync III-1988
 VDCompressDone III-1986
 VDReleaseCompressBuffer III-2030
 VDGetImageDescription III-2002
 VDResetCompressSequence III-2030
 VDSetCompressionOnOff III-2035
 VDGetCompressionTypes III-1994
 VDSetTimeBase III-2054
 VDSetFrameRate III-2041
 VDGetDataRate III-1997
 VDGetSoundInputDriver III-2019
 VDGetDMADepths III-1999
 VDGetPreferredTimeScale III-2017
 VDReleaseAsyncBuffers III-2029
 VDSetDataRate III-2037
 VDGetTimeCode III-2020
 VDUseSafeBuffers III-2056
 VDGetSoundInputSource III-2019
 VDGetCompressionTime III-1993
 VDSetPreferredPacketSize III-2052
 VDSetPreferredImageDimensions III-2051
 VDGetPreferredImageDimensions III-2016
 VDGetInputName III-2007
 VDSetDestinationPort III-2038
 SGInitialize III-1752
 SGSetDataOutput III-1785
 SGGetDataOutput III-1711
 SGSetGWorld III-1792
 SGGetGWorld III-1719
 SGNewChannel III-1753
 SGDisposeChannel III-1695
 SGStartPreview III-1818
 SGStartRecord III-1819
 SGIdle III-1751
 SGStop III-1819
 SGPause III-1771
 SGPrepare III-1772
 SGRelease III-1773
 SGGetMovie III-1724
 SGSetMaximumRecordTime III-1796
 SGGetMaximumRecordTime III-1723
 SGGetStorageSpaceRemaining III-1736
 SGGetTimeRemaining III-1739
 SGGrabPict III-1748

SGGetLastMovieResID III-1722
SGSetFlags III-1789
SGGetFlags III-1718
SGSetDataProc III-1787
SGNewChannelFromComponent III-1756
SGDisposeDeviceList III-1696
SGAppendDeviceListToMenu III-1685
SGSetSettings III-1801
SGGetSettings III-1731
SGGetIndChannel III-1720
SGUpdate III-1821
SGGetPause III-1730
SGSettingsDialog III-1808
SGGetAlignmentProc III-1698
SGSetChannelSettings III-1781
SGGetChannelSettings III-1707
SGGetMode III-1723
SGSetDataRef III-1787
SGGetDataRef III-1716
SGNewOutput III-1757
SGDisposeOutput III-1696
SGSetOutputFlags III-1797
SGSetChannelOutput III-1778
SGGetDataOutputStorageSpaceRemaining III-1713
SGHandleUpdateEvent III-1750
SGSetOutputNextOutput III-1800
SGGetOutputNextOutput III-1729
SGSetOutputMaximumOffset III-1799
SGGetOutputMaximumOffset III-1728
SGGetOutputDataReference III-1727
SGWriteExtendedMovieData III-1822
SGGetStorageSpaceRemaining64 III-1737
SGWriteMovieData III-1824
SGAddFrameReference III-1681
SGGetNextFrameReference III-1726
SGGetTimeBase III-1738
SGSortDeviceList III-1817
SGAddMovieData III-1682
SGChangedSource III-1686
SGAddExtendedFrameReference III-1677
SGGetNextExtendedFrameReference III-1725
SGAddExtendedMovieData III-1678
SGAddOutputDataRefToMedia III-1683
SGSetChannelUsage III-1782
SGGetChannelUsage III-1709
SGSetChannelBounds III-1774
SGGetChannelBounds III-1700
SGSetChannelVolume III-1783
SGGetChannelVolume III-1710

SGGetChannelInfo III-1702
 SGSetChannelPlayFlags III-1779
 SGGetChannelPlayFlags III-1705
 SGSetChannelMaxFrames III-1777
 SGGetChannelMaxFrames III-1704
 SGSetChannelRefCon III-1781
 SGSetChannelClip III-1775
 SGGetChannelClip III-1700
 SGGetChannelSampleDescription III-1706
 SGGetChannelDeviceList III-1701
 SGSetChannelDevice III-1776
 SGSetChannelMatrix III-1776
 SGGetChannelMatrix III-1703
 SGGetChannelTimeScale III-1708
 SGChannelPutPicture III-1689
 SGChannelSetRequestedDataRate III-1691
 SGChannelGetRequestedDataRate III-1688
 SGChannelSetDataSourceName III-1690
 SGChannelGetDataSourceName III-1687
 SGChannelSetCodecSettings III-1689
 SGChannelGetCodecSettings III-1686
 SGGetChannelTimeBase III-1708
 SGInitChannel III-1751
 SGWriteSamples III-1825
 SGGetDataRate III-1715
 SGAlignChannelRect III-1684
 SGPanelGetDitl III-1761
 SGPanelGetTitle III-1763
 SGPanelCanRun III-1759
 SGPanelInstall III-1764
 SGPanelEvent III-1760
 SGPanelItem III-1765
 SGPanelRemove III-1766
 SGPanelSetGrabber III-1767
 SGPanelSetResFile III-1768
 SGPanelGetSettings III-1762
 SGPanelSetSettings III-1769
 SGPanelValidateInput III-1770
 SGPanelSetEventFilter III-1767
 SGGetSrcVideoBounds III-1735
 SGSetVideoRect III-1816
 SGGetVideoRect III-1746
 SGGetVideoCompressorType III-1744
 SGSetVideoCompressorType III-1814
 SGSetVideoCompressor III-1812
 SGGetVideoCompressor III-1742
 SGGetVideoDigitizerComponent III-1745
 SGSetVideoDigitizerComponent III-1815
 SGVideoDigitizerChanged III-1822

SGSetVideoBottlenecks III-1811
SGGetVideoBottlenecks III-1741
SGGrabFrame III-1747
SGGrabFrameComplete III-1748
SGDisplayFrame III-1694
SGCompressFrame III-1692
SGCompressFrameComplete III-1692
SGAddFrame III-1680
SGTransferFrameForCompress III-1820
SGSetCompressBuffer III-1784
SGGetCompressBuffer III-1711
SGGetBufferInfo III-1699
SGSetUseScreenBuffer III-1811
SGGetUseScreenBuffer III-1740
SGGrabCompressComplete III-1746
SGDisplayCompress III-1693
SGSetFrameRate III-1792
SGGetFrameRate III-1719
SGSetPreferredPacketSize III-1801
SGGetPreferredPacketSize III-1730
SGSetUserVideoCompressorList III-1810
SGGetUserVideoCompressorList III-1740
SGSetSoundInputDriver III-1802
SGGetSoundInputDriver III-1732
SGSoundInputDriverChanged III-1817
SGSetSoundRecordChunkSize III-1805
SGGetSoundRecordChunkSize III-1734
SGSetSoundInputRate III-1804
SGGetSoundInputRate III-1734
SGSetSoundInputParameters III-1803
SGGetSoundInputParameters III-1733
SGSetAdditionalSoundRates III-1774
SGGetAdditionalSoundRates III-1697
SGSetFontName III-1790
SGSetFontSize III-1791
SGSetTextForeColor III-1807
SGSetTextBackColor III-1806
SGSetJustification III-1795
SGGetTextReturnToSpaceValue III-1737
SGSetTextReturnToSpaceValue III-1807
SGGetInstrument III-1721
SGSetInstrument III-1794
QTVideoOutputGetDisplayModeList II-1326
QTVideoOutputGetCurrentClientName II-1325
QTVideoOutputSetClientName II-1332
QTVideoOutputGetClientName II-1323
QTVideoOutputBegin II-1321
QTVideoOutputEnd II-1322
QTVideoOutputSetDisplayMode II-1332

QTVideoOutputGetDisplayMode II-1325
 QTVideoOutputCustomConfigureDisplay II-1322
 QTVideoOutputSaveState II-1331
 QTVideoOutputRestoreState II-1330
 QTVideoOutputGetGWorld II-1327
 QTVideoOutputGetGWorldParameters II-1328
 QTVideoOutputGetIndSoundOutput II-1329
 QTVideoOutputGetClock II-1324
 QTVideoOutputSetEchoPort II-1333
 NewDataHCompletionUPP II-1057
 NewVdigIntUPP II-1125
 NewSGDataUPP II-1106
 NewSGModalFilterUPP II-1109
 NewSGGrabBottleUPP II-1107
 NewSGGrabCompleteBottleUPP II-1108
 NewSGDisplayBottleUPP II-1106
 NewSGCompressBottleUPP II-1105
 NewSGCompressCompleteBottleUPP II-1105
 NewSGAddFrameBottleUPP II-1104
 NewSGTransferFrameBottleUPP II-1109
 NewSGGrabCompressCompleteBottleUPP II-1108
 NewSGDisplayCompressBottleUPP II-1107
 DisposeDataHCompletionUPP I-259
 DisposeVdigIntUPP I-303
 DisposeSGDataUPP I-290
 DisposeSGModalFilterUPP I-293
 DisposeSGGrabBottleUPP I-291
 DisposeSGGrabCompleteBottleUPP I-292
 DisposeSGDisplayBottleUPP I-290
 DisposeSGCompressBottleUPP I-289
 DisposeSGCompressCompleteBottleUPP I-289
 DisposeSGAddFrameBottleUPP I-289
 DisposeSGTransferFrameBottleUPP I-293
 DisposeSGGrabCompressCompleteBottleUPP I-292
 DisposeSGDisplayCompressBottleUPP I-291

QuickTimeMusic.h

QTMIDIGetMIDIPorts II-1188
 QTMIDIUseSendPort II-1190
 QTMIDISendMIDI II-1189
 QTMIDIUseReceivePort II-1189
 MusicGetDescription II-986
 MusicGetPart II-999
 MusicSetPart II-1010
 MusicSetPartInstrumentNumber II-1013
 MusicGetPartInstrumentNumber II-1002
 MusicStorePartInstrument II-1017

MusicGetPartAtomicInstrument II-1000
MusicSetPartAtomicInstrument II-1011
MusicGetInstrumentKnobDescriptionObsolete II-992
MusicGetDrumKnobDescriptionObsolete II-988
MusicGetKnobDescriptionObsolete II-995
MusicGetPartKnob II-1002
MusicSetPartKnob II-1014
MusicGetKnob II-994
MusicSetKnob II-1007
MusicGetPartName II-1003
MusicSetPartName II-1015
MusicFindTone II-981
MusicPlayNote II-1004
MusicResetPart II-1006
MusicSetPartController II-1012
MusicGetPartController II-1001
MusicGetMIDIProc II-998
MusicSetMIDIProc II-1009
MusicGetInstrumentNames II-993
MusicGetDrumNames II-989
MusicGetMasterTune II-997
MusicSetMasterTune II-1008
MusicGetInstrumentAboutInfo II-990
MusicGetDeviceConnection II-987
MusicUseDeviceConnection II-1019
MusicGetKnobSettingStrings II-996
MusicGetMIDIPorts II-997
MusicSendMIDI II-1006
MusicReceiveMIDI II-1005
MusicStartOffline II-1016
MusicSetOfflineTimeTo II-1009
MusicGetInstrumentKnobDescription II-992
MusicGetDrumKnobDescription II-988
MusicGetKnobDescription II-995
MusicGetInfoText II-990
MusicGetInstrumentInfo II-991
MusicTask II-1018
MusicSetPartInstrumentNumberInterruptSafe II-1013
MusicSetPartSoundLocalization II-1015
MusicGenericConfigure II-982
MusicGenericGetPart II-985
MusicGenericGetKnobList II-984
MusicGenericSetResourceNumbers II-986
MusicDerivedMIDISend II-975
MusicDerivedSetKnob II-976
MusicDerivedSetPart II-979
MusicDerivedSetInstrument II-976
MusicDerivedSetPartInstrumentNumber II-979
MusicDerivedSetMIDI II-978

MusicDerivedStorePartInstrument II-980
 MusicDerivedOpenResFile II-975
 MusicDerivedCloseResFile II-974
 InstrumentGetInst II-767
 InstrumentGetInfo II-766
 InstrumentInitialize II-769
 InstrumentOpenComponentResFile II-770
 InstrumentCloseComponentResFile II-765
 InstrumentGetComponentRefCon II-766
 InstrumentSetComponentRefCon II-771
 InstrumentGetSynthesizerType II-768
 NARegisterMusicDevice II-1038
 NAUnregisterMusicDevice II-1051
 NAGetRegisteredMusicDevice II-1028
 NASaveMusicConfiguration II-1039
 NANewNoteChannel II-1030
 NADisposeNoteChannel II-1020
 NAGetNoteChannelInfo II-1026
 NAPrereollNoteChannel II-1037
 NAUnrollNoteChannel II-1051
 NASetNoteChannelVolume II-1047
 NAResetNoteChannel II-1039
 NAPlayNote II-1036
 NASetController II-1042
 NASetKnob II-1045
 NAFindNoteChannelTone II-1021
 NASetInstrumentNumber II-1044
 NAPickInstrument II-1035
 NAPickArrangement II-1032
 NASetDefaultMIDIInput II-1043
 NAGetDefaultMIDIInput II-1023
 NAUseDefaultMIDIInput II-1052
 NALoseDefaultMIDIInput II-1029
 NASTuffToneDescription II-1048
 NACopyrightDialog II-1019
 NAGetIndNoteChannel II-1023
 NAGetMIDIPorts II-1025
 NAGetNoteRequest II-1027
 NASendMIDI II-1040
 NAPickEditInstrument II-1033
 NANewNoteChannelFromAtomicInstrument II-1031
 NASetAtomicInstrument II-1041
 NAGetKnob II-1024
 NATask II-1049
 NASetNoteChannelBalance II-1046
 NASetInstrumentNumberInterruptSafe II-1044
 NASetNoteChannelSoundLocalization II-1047
 NAGetController II-1022
 TuneSetHeader III-1967

TuneGetTimeBase III-1961
TuneSetTimeScale III-1973
TuneGetTimeScale III-1962
TuneGetIndexedNoteChannel III-1958
TuneQueue III-1964
TuneInstant III-1963
TuneGetStatus III-1961
TuneStop III-1975
TuneSetVolume III-1974
TuneGetVolume III-1963
TunePreroll III-1964
TuneUnroll III-1976
TuneSetNoteChannels III-1969
TuneSetPartTranspose III-1971
TuneGetNoteAllocator III-1959
TuneSetSofter III-1972
TuneTask III-1975
TuneSetBalance III-1966
TuneSetSoundLocalization III-1972
TuneSetHeaderWithSize III-1968
TuneSetPartMix III-1970
TuneGetPartMix III-1960
NewMusicMIDISendUPP II-1094
NewMusicMIDIReadHookUPP II-1093
NewMusicOfflineDataUPP II-1094
NewTuneCallBackUPP II-1122
NewTunePlayCallBackUPP II-1123
DisposeMusicMIDISendUPP I-280
DisposeMusicMIDIReadHookUPP I-279
DisposeMusicOfflineDataUPP I-280
DisposeTuneCallBackUPP I-302
DisposeTunePlayCallBackUPP I-302

QuickTimeStreaming.h

InitializeQTS II-758
TerminateQTS III-1927
QTSNewPresentation II-1250
QTSDisposePresentation II-1221
QTSPresIdle II-1284
QTSPresInvalidateRegion II-1284
QTSPresSetFlags II-1291
QTSPresGetFlags II-1268
QTSPresGetTimeBase II-1281
QTSPresGetTimeScale II-1282
QTSPresSetInfo II-1293
QTSPresGetInfo II-1272
QTSPresHasCharacteristic II-1283

QTSPresSetNotificationProc II-1296
 QTSPresGetNotificationProc II-1275
 QTSPresPreroll II-1285
 QTSPresPreroll64 II-1286
 QTSPresStart II-1303
 QTSPresSkipTo II-1302
 QTSPresSkipTo64 II-1302
 QTSPresStop II-1304
 QTSPresNewStream II-1284
 QTSDisposeStream II-1222
 QTSPresGetNumStreams II-1277
 QTSPresGetIndStream II-1271
 QTSGetStreamPresentation II-1239
 QTSPresSetPreferredRate II-1297
 QTSPresGetPreferredRate II-1278
 QTSPresSetEnable II-1290
 QTSPresGetEnable II-1267
 QTSPresSetPresenting II-1298
 QTSPresGetPresenting II-1279
 QTSPresSetActiveSegment II-1288
 QTSPresGetActiveSegment II-1265
 QTSPresSetPlayHints II-1296
 QTSPresGetPlayHints II-1278
 QTSPresSetGWorld II-1292
 QTSPresGetGWorld II-1270
 QTSPresSetClip II-1289
 QTSPresGetClip II-1266
 QTSPresSetMatrix II-1295
 QTSPresGetMatrix II-1275
 QTSPresSetDimensions II-1290
 QTSPresGetDimensions II-1266
 QTSPresSetGraphicsMode II-1292
 QTSPresGetGraphicsMode II-1269
 QTSPresGetPicture II-1277
 QTSPresSetVolumes II-1301
 QTSPresGetVolumes II-1282
 QTSSetNetworkAppName II-1305
 QTSGetNetworkAppName II-1237
 QTSNewStatHelper II-1255
 QTSDisposeStatHelper II-1221
 QTSStatHelperGetStats II-1310
 QTSStatHelperResetIter II-1312
 QTSStatHelperNext II-1311
 QTSStatHelperGetNumStats II-1310
 QTSGetOrMakeStatAtomForStream II-1238
 QTSInsertStatistic II-1239
 QTSInsertStatisticName II-1241
 QTSInsertStatisticUnits II-1242
 QTSPrefsAddProxySetting II-1257

QTSPrefsFindProxyByType II-1260
 QTSPrefsAddConnectionSetting II-1257
 QTSPrefsFindConnectionByType II-1259
 QTSPrefsGetActiveConnection II-1262
 QTSPrefsGetNoProxyURLs II-1263
 QTSPrefsSetNoProxyURLs II-1263
 QTSNewPtr II-1253
 QTSNewHandle II-1249
 QTSAAllocMemPtr II-1218
 QTSReleaseMemPtr II-1304
 QTSAAllocBuffer II-1218
 QTSFreeMessage II-1236
 QTSDupMessage II-1223
 QTSCopyMessage II-1220
 QTSFlattenMessage II-1235
 QTSMessageLength II-1249
 QTSGetErrorString II-1236
 NewQTSNotificationUPP II-1098
 DisposeQTSNotificationUPP I-283

QuickTimeVR.h

NewQTVRLeavingNodeUPP II-1101
 NewQTVREnteringNodeUPP II-1100
 NewQTVRMouseOverHotSpotUPP II-1102
 NewQTVRImagingCompleteUPP II-1100
 NewQTVRBackBufferImagingUPP II-1099
 DisposeQTVRLeavingNodeUPP I-286
 DisposeQTVREnteringNodeUPP I-285
 DisposeQTVRMouseOverHotSpotUPP I-286
 DisposeQTVRImagingCompleteUPP I-285
 DisposeQTVRBackBufferImagingUPP I-284
 NewQTVRInterceptUPP II-1101
 DisposeQTVRInterceptUPP I-286
 InitializeQTVR II-758
 TerminateQTVR III-1927
 QTVRGetQTVRTrack II-1375
 QTVRGetQTVRInstance II-1373
 QTVRSetPanAngle II-1431
 QTVRGetPanAngle II-1373
 QTVRSetTiltAngle II-1434
 QTVRGetTiltAngle II-1377
 QTVRSetFieldOfView II-1420
 QTVRGetFieldOfView II-1360
 QTVRShowDefaultView II-1442
 QTVRSetViewCenter II-1437
 QTVRGetViewCenter II-1378
 QTVRNudge II-1402

QTVRInteractionNudge II-1389
 QTVRGetVRWorld II-1385
 QTVRGetNodeInfo II-1371
 QTVRGoToNodeID II-1386
 QTVRGetCurrentNodeID II-1359
 QTVRGetNodeType II-1372
 QTVRPtToHotSpotID II-1405
 QTVRGetHotSpotType II-1364
 QTVRTriggerHotSpot II-1443
 QTVRSetMouseOverHotSpotProc II-1429
 QTVREnableHotSpot II-1340
 QTVRGetVisibleHotSpots II-1384
 QTVRGetHotSpotRegion II-1363
 QTVRSetMouseOverTracking II-1431
 QTVRGetMouseOverTracking II-1370
 QTVRSetMouseDownTracking II-1429
 QTVRGetMouseDownTracking II-1370
 QTVRMouseEnter II-1392
 QTVRMouseWithin II-1401
 QTVRMouseLeave II-1394
 QTVRMouseDown II-1391
 QTVRMouseStillDown II-1395
 QTVRMouseUp II-1398
 QTVRMouseStillDownExtended II-1396
 QTVRMouseUpExtended II-1399
 QTVRInstallInterceptProc II-1387
 QTVRCallInterceptedProc II-1336
 QTVRGetCurrentMouseMode II-1358
 QTVRSetFrameRate II-1421
 QTVRGetFrameRate II-1362
 QTVRSetViewRate II-1439
 QTVRGetViewRate II-1381
 QTVRSetViewCurrentTime II-1438
 QTVRGetViewCurrentTime II-1379
 QTVRGetCurrentViewDuration II-1360
 QTVRSetViewState II-1440
 QTVRGetViewState II-1382
 QTVRGetViewStateCount II-1383
 QTVRSetAnimationSetting II-1409
 QTVRGetAnimationSetting II-1344
 QTVRSetControlSetting II-1416
 QTVRGetControlSetting II-1355
 QTVREnableFrameAnimation II-1339
 QTVRGetFrameAnimation II-1361
 QTVREnableViewAnimation II-1342
 QTVRGetViewAnimation II-1377
 QTVRSetVisible II-1441
 QTVRGetVisible II-1384
 QTVRSetImagingProperty II-1422

QTVRGetImagingProperty II-1365
QTVRUpdate II-1445
QTVRBeginUpdateStream II-1335
QTVREndUpdateStream II-1343
QTVRSetTransitionProperty II-1435
QTVREnableTransition II-1341
QTVRSetAngularUnits II-1408
QTVRGetAngularUnits II-1344
QTVRPtToAngles II-1404
QTVRCoordToAngles II-1338
QTVRAnglesToCoord II-1334
QTVRPanToColumn II-1403
QTVRColumnToPan II-1337
QTVRTiltToRow II-1443
QTVRRowToTilt II-1407
QTVRWrapAndConstrain II-1446
QTVRSetEnteringNodeProc II-1419
QTVRSetLeavingNodeProc II-1428
QTVRSetInteractionProperty II-1425
QTVRGetInteractionProperty II-1368
QTVRReplaceCursor II-1407
QTVRGetViewingLimits II-1379
QTVRGetConstraintStatus II-1353
QTVRGetConstraints II-1352
QTVRSetConstraints II-1415
QTVRGetAvailableResolutions II-1346
QTVRGetBackBufferMemInfo II-1347
QTVRGetBackBufferSettings II-1350
QTVRSetBackBufferPrefs II-1412
QTVRSetPrescreenImagingCompleteProc II-1432
QTVRSetBackBufferImagingProc II-1411
QTVRRefreshBackBuffer II-1406

QuickTimeVRFormat.h

No functions are declared in QuickTimeVRFormat.h.

Sound.h

NewSndCallBackUPP II-1111
DisposeSndCallBackUPP I-294
NewSndDoubleBackUPP II-1111
DisposeSndDoubleBackUPP I-295
NewSoundParamUPP II-1112
NewSoundConverterFillBufferDataUPP II-1112
NewSIInterruptUPP II-1110

NewSICompletionUPP II-1110
 DisposeSoundParamUPP I-296
 DisposeSoundConverterFillBufferDataUPP I-295
 DisposeSIInterruptUPP I-294
 DisposeSICompletionUPP I-293
 NewFilePlayCompletionUPP II-1059
 DisposeFilePlayCompletionUPP I-261
 SysBeep III-1915
 SndDoCommand III-1831
 SndDoImmediate III-1832
 SndNewChannel III-1839
 SndDisposeChannel III-1830
 SndPlay III-1842
 SndAddModifier III-1828
 SndControl III-1829
 SndSoundManagerVersion III-1847
 SndStartFilePlay III-1847
 SndPauseFilePlay III-1841
 SndStopFilePlay III-1848
 SndChannelStatus III-1829
 SndManagerStatus III-1839
 SndGetSysBeepState III-1833
 SndSetSysBeepState III-1846
 SndPlayDoubleBuffer III-1843
 MACEVersion II-784
 Comp3to1 I-104
 Explto3 I-342
 Comp6to1 I-106
 Explto6 I-343
 GetSysBeepVolume I-514
 SetSysBeepVolume III-1647
 GetDefaultOutputVolume I-408
 SetDefaultOutputVolume III-1582
 GetSoundHeaderOffset I-510
 UnsignedFixedMulDiv III-1980
 GetCompressionInfo I-398
 SetSoundPreference III-1641
 GetSoundPreference I-512
 OpenMixerSoundComponent II-1132
 CloseMixerSoundComponent I-102
 SndGetInfo III-1833
 SndSetInfo III-1845
 GetSoundOutputInfo I-511
 SetSoundOutputInfo III-1640
 GetCompressionName I-400
 SoundConverterOpen III-1867
 SoundConverterClose III-1861
 SoundConverterGetBufferSizes III-1866
 SoundConverterBeginConversion III-1861

SoundConverterConvertBuffer III-1862
SoundConverterEndConversion III-1863
SoundConverterGetInfo III-1867
SoundConverterSetInfo III-1868
SoundConverterFillBuffer III-1864
SoundManagerGetInfo III-1869
SoundManagerSetInfo III-1870
SoundComponentInitOutputDevice III-1852
SoundComponentSetSource III-1858
SoundComponentGetSource III-1850
SoundComponentGetSourceData III-1851
SoundComponentSetOutput III-1857
SoundComponentAddSource III-1848
SoundComponentRemoveSource III-1855
SoundComponentGetInfo III-1849
SoundComponentSetInfo III-1856
SoundComponentStartSource III-1859
SoundComponentStopSource III-1860
SoundComponentPauseSource III-1853
SoundComponentPlaySourceBuffer III-1854
AudioGetVolume I-49
AudioSetVolume I-52
AudioGetMute I-48
AudioSetMute I-51
AudioSetToDefaults I-51
AudioGetInfo I-48
AudioGetBass I-47
AudioSetBass I-50
AudioGetTreble I-49
AudioSetTreble I-51
AudioGetOutputDevice I-49
AudioMuteOnEvent I-50
SPBVersion III-1884
SndRecord III-1843
SndRecordToFile III-1845
SPBSignInDevice III-1882
SPBSignOutDevice III-1883
SPBGetIndexedDevice III-1873
SPBOpenDevice III-1876
SPBCloseDevice III-1871
SPBRecord III-1878
SPBRecordToFile III-1880
SPBPauseRecording III-1877
SPBResumeRecording III-1881
SPBStopRecording III-1883
SPBGetRecordingStatus III-1874
SPBGetDeviceInfo III-1871
SPBSetDeviceInfo III-1881
SPBMillisecondsToBytes III-1875

SPBBytesToMilliseconds III-1870
SetupSndHeader III-1672
SetupAIFFHeader III-1670
ParseAIFFHeader II-1136
ParseSndHeader II-1136
SndInputReadAsync III-1836
SndInputReadSync III-1837
SndInputPauseRecording III-1836
SndInputResumeRecording III-1837
SndInputStopRecording III-1838
SndInputGetStatus III-1835
SndInputGetDeviceInfo III-1834
SndInputSetDeviceInfo III-1838
SndInputInitHardware III-1835

Bibliography

All the volumes of QuickTime API developer documentation are available online for download from Apple's QuickTime Technical Publications website at <http://developer.apple.com/techpubs/quicktime/qtdevdocs/RM/frameset2.htm>

This website is the most current and up-to-date source for all QuickTime developer documentation. A complete roadmap of topics and functions is provided for developers who want to build applications using the QuickTime API. The QuickTime technical documentation suite totals more than 7,000 pages.

QuickTime Programming Books in PDF

QuickTime developer documents are also available in Adobe Portable Document format (PDF). PDF files can be opened and viewed online, as well as downloaded from

<http://developer.apple.com/techpubs/quicktime/qtdevdocs/RM/PDF.htm>

From this site you can download the following books, cited in this reference:

- *Inside Macintosh: QuickTime*
- *Inside Macintosh: QuickTime Components*
- *Programming with QuickTime VR 2.1*
- *Inside QuickTime: QuickTime File Format*

Other useful books that you can download include:

- *What's New in QuickTime 4.1*: documents the new features of QuickTime 4.1.
- *Mac OS for QuickTime Programmers*: documents the Mac OS system calls and data structures that are included in QuickTime 4 for Windows and used by QuickTime developers. This material is also documented in various volumes of Inside Macintosh. It is brought together here primarily for the convenience of Windows programmers.
- *QuickTime Wired Movies And Sprite Animation*: Describes and documents the API for creating interactive movies and sprites, including HREF Tracks and wired sprites.

- *QuickTime Streaming*: Documents the features and API for QuickTime Streaming over RTP.
- *QTSS Streaming Server API*: Documents the API for adding modules to the QuickTime Streaming Server.
- *QuickTime For Java Summary*: An overview of and introduction to the QuickTime for Java API.
- *QuickTime for Windows Programmers*: Describes the features of QuickTime programming that are unique to Windows.
- *QuickTime Music Architecture for Macintosh and Windows*: Provides information for Mac and Windows developers who are adding QuickTime music support to their applications.
- *Mac OS Sound, Including Sound Manager 3.3*: Documents all aspects of using sound for QuickTime developers.
- *3D Graphics Programming with QuickDraw 3D*: Programming guide to QuickDraw 3D, version 1.5.4
- *Improvements in QuickDraw 3D 1.6*: Documents changes and additions to QuickDraw 3D, version 1.6. You also need to read *3D Graphics Programming with QuickDraw 3D*.
- *Making Cool QuickDraw 3D Applications*: Programming hints for 3D developers.

The QuickTime Developer Series

Three overview books are currently available in the QuickTime Developer Series. These books are published by Morgan Kaufmann; they are available from online booksellers and most computer bookstores.

- *Discovering QuickTime* (ISBN 0-12-059640-7), 515 pp. This is the official guide to programming QuickTime in C. It includes working demos and code samples on its CD-ROM.
- *QuickTime for Java* (ISBN 0-12-305440-0), 655 pp. This is the official guide for Java programmers who want to use QuickTime. It includes a full description of the Java interface to QuickTime and a CD-ROM with working examples.
- *QuickTime for the Web* (ISBN 0-12-471255-X), 720 pp. This is the most comprehensive book available for authoring QuickTime movies on the Web. The CD includes QuickTime 4 Pro for both Macintosh and Windows.

Inside Macintosh

The original 25 volumes of Inside Macintosh, covering all aspects of Mac OS programming, were published by Addison-Wesley. The following books, cited in this reference, are available online at

<http://developer.apple.com/techpubs/macos8/Indexes/documentindex.html>:

- Advanced Color Imaging on the Mac OS
- Files
- Imaging With QuickDraw
- Memory
- More Macintosh Toolbox
- PowerPC System Software
- Sound
- Text

Of these books, the following titles are available in printed editions from

<http://www.fatbrain.com/>:

- Imaging With QuickDraw
- More Macintosh Toolbox

Some Useful QuickTime Websites

- Here is Apple's official site for information, demos, sample code, online documentation, and the latest software:

<http://www.apple.com/quicktime/>

- QuickTime Live! is an Apple-sponsored conference for both content providers and Macintosh and Windows application developers:

<http://www.apple.com/quicktimelive/>

- QuickTime terms and definitions current in the industry, plus a good set of links to other QuickTime sites:

<http://www.pcwebopedia.com/QuickTime.htm>

- Channel QuickTime, an Apple home page with links to FAQs, forums, and the latest news about QuickTime:

<http://www.quicktimefaq.org/>

B I B L I O G R A P H Y

- The entry point for a variety of announcements and discussion forums about QuickTime, multimedia, and other topics of interest to QuickTime developers:

<http://www.lists.apple.com/>

- A site with lots of goodies, maintained by the authors of the MoviePlayer documentation and updated frequently. A useful resource:

<http://www.bmug.org/quicktime/>

- The International QuickTime VR Association website, a professional association that promotes and supports the use of QuickTime VR and related technologies worldwide:

<http://www.iqtvra.org/>

Glossary

alias

An object in the Mac OS file system that represents a file, directory, or volume.

band

A horizontal strip from an image. QuickTime may break an image into bands if a codec cannot process the whole image at one time.

base graphics exporter

A component that handles most of the tasks of a graphics exporter. You can build on the base component to reduce the amount of work required to create a custom graphics exporter.

base graphics importer

A component that handles most of the tasks of a graphics importer. You can build on the base component to reduce the amount of work required to create a custom graphics importer.

big-endian

Data formatting in which multibyte fields are addressed by pointing to their most significant bytes.

blacklining

A technique in which every other horizontal line of an image is displayed.

callback event

A scheduled invocation of a callback function. Your application establishes the criteria that determine when the callback is to be invoked.

CarbonLib

The main API library for Carbon, the currently-supported Mac OS system software.

color lookup table

A data structure that maps QuickDraw color indexes into actual RGB color values.

dither

To mix existing colors together, by alternating their pixels, creating the illusion of a third color that is otherwise unavailable.

Glossary

HFS

Hierarchical File System, the file system used in current versions of the Mac OS.

interlace

A video mode that updates alternate scan lines on succeeding screen refreshes.

key frame

A sample in a sequence of temporally compressed samples that does not rely on other samples for any of its information. Also called a sync sample.

little-endian

Data formatting in which multibyte fields are addressed by pointing to their least significant bytes.

MACE

Macintosh Audio Compression and Expansion, an audio compression format that QuickTime supports but which is not recommended for new applications.

blend matte

A pixel map that defines the blending of video and digital data for a video digitizer component. The value of each pixel in the matte determines the relative intensity of the corresponding video pixel in the resulting image.

modifier key

In Macintosh computers, any of the Option, Command, Caps Lock, Control, or Shift keys.

picture file

A file of QuickDraw drawing commands.

poster

A frame shot from a QuickTime movie that is used to represent its content to the user.

preview

A short, potentially dynamic, visual representation of the contents of a file. File dialog boxes present previews to the user.

progress function

An application-defined function that you can use to track the progress of an operation and keep the user informed.

region code

A constant that defines a geographical region for purposes of localization.

resource

In Mac OS programming, any data stored as a defined structure in the resource fork of a Macintosh file. The data in the resource is interpreted according to its resource type.

shadow sync

A self-contained sample that is an alternate for an already existing frame difference sample. During certain random-access operations, a shadow sync sample is used instead of a normal key frame, which may be very far away from the desired frame.

sprite

An animated image that is managed by QuickTime. A sprite is defined once and is then animated by commands that change its position or appearance.

Standard File Package

Part of the Mac OS system software that manages the user interface for naming, choosing, and identifying files.

sticky error

One of two error values maintained by the Movie Toolbox. The sticky error is updated only when an application directs the Movie Toolbox to do so. The other error value, the current error, is updated by every Movie Toolbox function.

sync sample

A sample in a sequence of temporally compressed samples that does not rely on other samples for any of its information. Also called a key frame.

32-bit clean

A programming environment in which the OS manages all bits of 32-bit addresses and applications do not use any high bits for non-addressing purposes.

ticks

Sixtieths of a second. QuickTime often measures time intervals in ticks.

time structure

A TimeRecord data structure, which contains a time value, a time scale in units per second, and a reference to a time base.

tween

To interpolate new data between given values in conformance to an algorithm. It is an efficient way to expand or smooth a movie's presentation between its actual frames.

Glossary

Unicode

An international standard that encodes written language marks as 16-bit values. It covers Chinese, Japanese, and Korean characters plus all the world's current alphabets.

Universal Procedure Pointer

In the Mac OS, a 680x0 procedure pointer or the address of a routine descriptor that can route calls to a PowerPC processor.

user data list

A QuickTime data structure that contains readable text information about a movie.

vector data stream

An ordered series of QuickTime atoms that contain information about vector paths and their characteristics.

Document History

This is the first issue of this document.

DOCUMENT HISTORY